Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Síťová virtualizace

Network virtualization

2017/2018 **Bc. Petr Šupka**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Petr Šupka**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2601T013 Telekomunikační technika

Téma: Síťová virtualizace
Network virtualization

Jazyk vypracování: čeština

Zásady pro vypracování:

Nedílnou součástí virtualizace je i virtualizace sítí, kde se uplatňují technologie VEB - Virtual Ethernet Bridge podle standardu IEEE 802.1-2014. Technologie VEB zajišťuje jednotlivým virtuálním systémům přístup do sítě s různými vlastnostmi. Další progresivní technologii je IPv6 - Internet protokol verze 6. Cílem diplomové práce je vytvoření virtuálních hostů s různými virtuálními rozhraními s protokolem IPv6, testování jejich vlastností a výkonnostní porovnání.

Diplomová práce bude obsahovat
- Popis základních vlastností technologie VEB
- Popis uplatnění technologie VEB v operačním systému Linux
- Testovací nástroje vlastností a výkonnostní testovaní
- Vyhodnocení testování

Seznam doporučené odborné literatury:

- IEEE 802.1Q-2014 - Bridges and Bridged Networks
- Gregory Boyce: Linux Networking Cookbook, Packt Publishing, 2016, ISBN-13: 978-1785287916
- Paul Cobbaut: Mastering Linux - Networking, Samurai Media Limited, 2016, ISBN-13: 978-9888406210
- Sachidananda Kangovi: Peering Carrier Ethernet Networks, Morgan Kaufmann, 2016, ISBN-13: 978-0128053195

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Ing. Jaroslav Zdrálek, Ph.D.**

Datum zadání: 01.09.2016
Datum odevzdání: 30.04.2018

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * 1D16 * OSTRAVA

doc. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostrožské Lhotě dne: *30. dubna 2018*

podpis studenta

Poděkování

Rád bych poděkoval doc. Ing. Jaroslavu Zdrálkovi, Ph.D za odbornou pomoc a konzultaci při vytváření této diplomové práce.

Abstrakt

Diplomová práce se zabývá virtualizací síťových rozhraní, která jsou dnes běžně používána ve světě serverů. Tato rozhraní se pomalu přesunují z IPv4 protokolu na IPv6 protokol. V první části této práce je objasněna teoretická část technologií VEB (Virtual Ethernet Bridging). Zároveň práce popisuje možné způsoby zapojení a konfigurace těchto virtuálních rozhraní. Dále pojednává o problematice IPv6 protokolu a jeho využití v počítačových sítích. V praktické části diplomové práce budu zabývat samotnou konfigurací těchto rozhraní na virtuální počítače, takzvané kontejnery, běžící pod operačním systémem Linux..

Vycházel jsem z problematiky virtualizace síťových rozhraní pomocí IPv6 protokolu. Mou snahou bylo ověřit nejvhodnější síťové nastavení virtuálních strojů pro užití s IPv6 protokolem. Chtěl jsem navrhnout řešení pro připojení z vnější LAN sítě na virtualizované stroje s virtualizovanými síťovými kartami za fyzickou síťovou kartou. K tomu jsem musel provést testování několika různých způsobů přístupu k síťové virtualizaci.

Klíčová slova

Linux, virtualizace; VEB; fyzická síťová karta; virutální síťová karta; hypervisor; kontejner; host; přepínač; směrovač;

Abstract

This thesis deals with network interface virtualization. Nowadays, these interfaces are commonly used in server techhnologies. These interfaces are moving from IPv4 protocol to IPv6 protocol. The first part of this thesis is theoretical part of Virtual Ethernet Bridging technology. I will also explain different ways of attach and configuration of these virtual interfaces. Another part explains the possibility of using Linux containers as they become a competition to typical virtualizaton of whole operating system. The practical part of this thesis will cover configuration of virtual interfaces on virtual containers using Linux operating system.

My effort is to check different network settings for virtual stations using IPv6 protocol. I would like to suggest appropriate solution for capability of connecting from outside LAN to virtual stations with virtual network interface cards, which are virtual stations behind the physical network card. I will have to do a several tests of different approaches to network virtualization.

Key words

Linux, virtualization; VEB, physical network card; virtual network card; hypervisor; container; host; switch; router;

Seznam použitých zkratek

Zkratka	Význam
API	Application Programming Interface
CIDR	Classless Inter-Domain Routing
CLI	Command Line Interface
DHCP	Dynamic Host Configuration Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
LXC	Linux Containers
NIC	Network Interface Card
VEB	Virtual Ethernet Bridging
VEPA	Virtual Ethernet Port Aggregator
VM	Virtual Machine
vNIC	Virtual Network Interface Card
VSI	Virtual Station Interface

Obsah

Úvod

V dnešní době neustále větších nároků na fyzický hardware serverů a aktivních síťových prvků je potřeba hledat řešení, která by zajistila menší požadavky na tyto zařízení. Spousta institucí již přešla na virtualizované operační systémy pomocí hypervisorů a jejich schopnost vytvářet virtuální sítě. Bohužel, hypervisory jsou náročné na samotný hardware a každá virtuální stanice, která je vytvořena v hypervisoru zbytečně obsahuje plnohodnotná operační systém. Samozřejmě při nedostatečném výkonu serveru je možno takové zařízení modernizovat pomocí koupě nového hardwaru, ale ne vždy jsou investice jediným řešením.

Jako řešení na tento problém jsem se rozhodl vytvořit virtuální kontejnery, které jsou mnohem méně náročné na požadavky výkonu fyzického stroje, protože ke svému vytvoření nepotřebují instalaci plnohodnotného operačního systému. Zároveň se v práci budu zabývat technologiemi k vytváření virtuálních sítí, které mají různé nastavení a lze je použít s různými požadavky na takovou síť. Díky tomu bychom měli být schopni získat virtuální sítě, které chováním odpovídají sítím fyzickým, ale nejsou limitovány rychlostmi fyzických síťových karet.

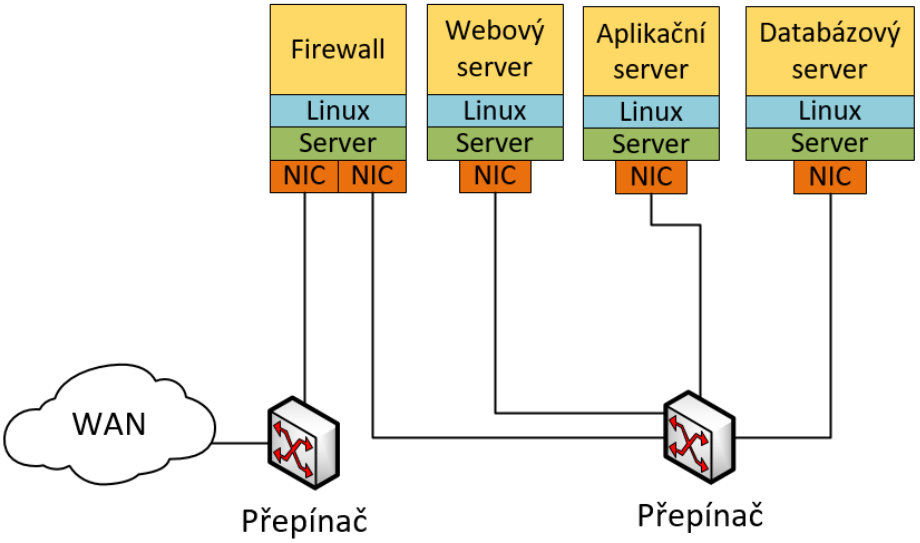
V práci se budu zabývat zkoumáním virtualizace kontejnerů pomocí LXC a taktéž konfigurací virtuálních sítí s různými nastaveními. Každé nastavení sítě totiž funguje odlišně a já se budu snažit najít způsoby pro efektivní využití těchto nastavení ke konkrétním případům. V práci se pokusím vysvětlit možnosti přenosu virtuálních strojů pomocí VEB technologie, která se dnes používá při síťové správě virtuálních stanic.

V praktické části se budu zabývat propustnostmi síťových a virtuálních síťových rozhraní, protože očekávám, že přenosové rychlosti virtuálních síťových karet budou rychlejší, než těch fyzických. Dále budu zkoumat reálnou cestu dat přes síťová rozhraní a vyhodnotím, jestli je tento způsob přínosem, nebo ne.

Cílem celé práce je vytvoření virtuálního stroje s několika kontejnery, které budou mít různá síťová nastavení a se správnou síťovou konfigurací budou dostupné z lokální sítě a internetu. Díky tomu bychom měli být schopni vytvořit například prostředí pro uživatele, kteří potřebují třeba jenom jednu službu dostupnou ze sítě. V takovém případě je tedy zbytečné instalovat kompletně celou virtuální stanici. Proto se budu snažit ukázat a vysvětlit, proč jsou kontejnery velmi dobrou cestou k virtualizaci a jak lze tyto kontejnery spravovat pomocí virtuální sítě.

1 Virtualizace sítí

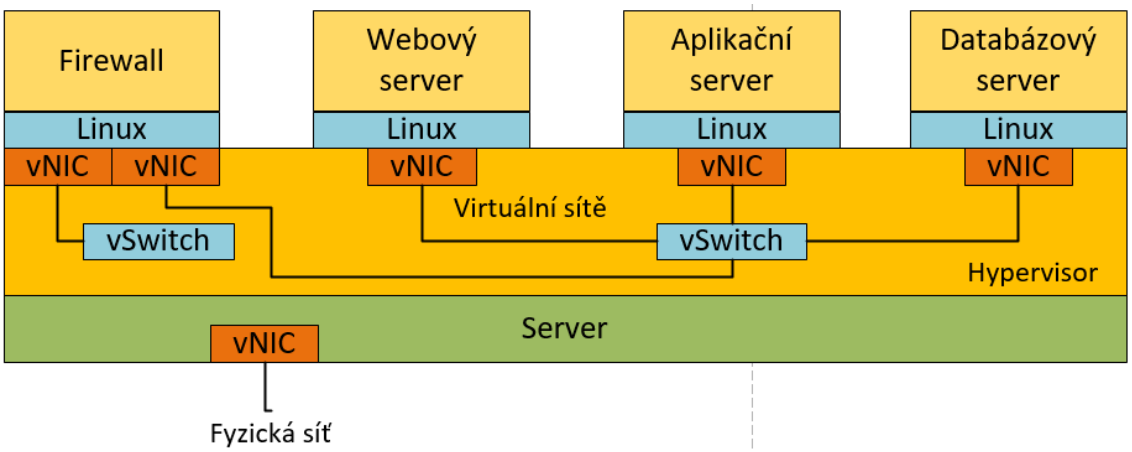
Na obrázku 1.1 je v obvyklém prostředí množina fyzických serverů, která hostuje nezbytné soubory aplikací. Pro možnost komunikace mezi servery, je v každém z těchto serverů jedna nebo více síťových karet, které jsou připojeny do externí síťové infrastruktury. Síťová karta spolu se síťovým softwarovým balíkem umožňuje mezi koncovými body skrze síťovou infrastrukturu. Jak je vidět, tato funkčnost je zajištěna pomocí přepínače, který umožňuje efektivní komunikaci mezi jednotlivými koncovými body.



Obrázek 1.1: *Obvyklé prostředí s více fyzickými servery*

Klíčovou inovací v konsolidaci serverů je možnost sdílení fyzického hardwaru pro více operačních systémů a aplikací (obrázek 1.2). Tato inovace se nazývá hypervisor. Každý virtuální stroj nahlíží na hardware, na kterém běží, jako na nevyužitý zdroj výpočetního výkonu a to i přesto, že reálně je výpočetní výkon využíván i jinými virtuálními stroji. Jako příklad tohoto chování lze uvést například virtuální síťovou kartu.

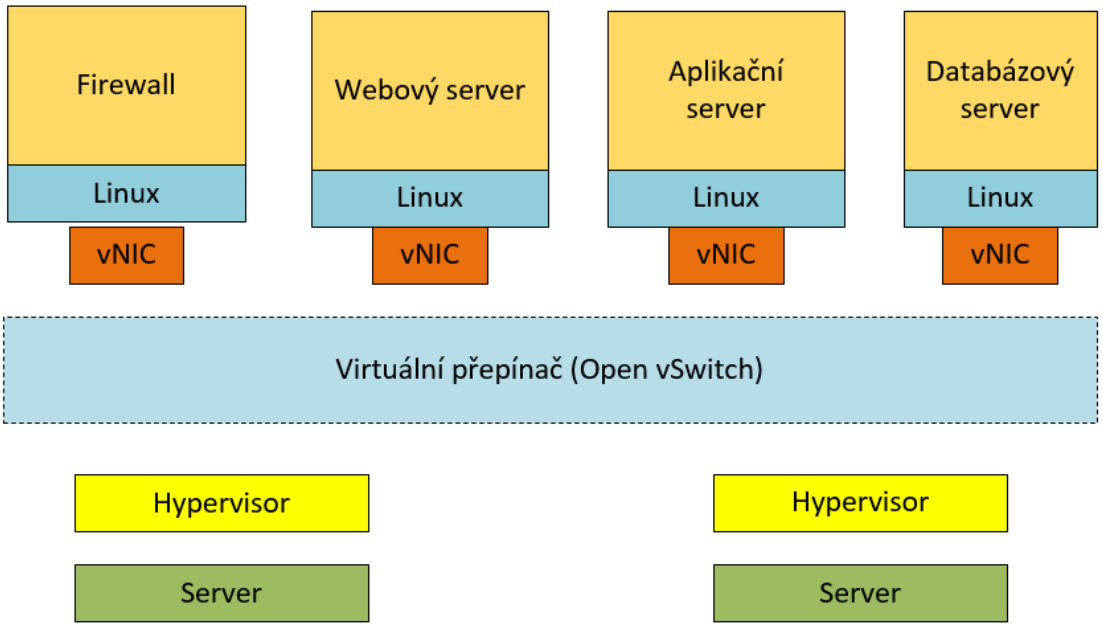
Hypervisor může vytvořit jednu nebo více těchto virtuálních síťových karet pro každý virtuální stroj. Tyto síťové karty se mohou virtuálnímu stroji jevit jako fyzické, ale ve skutečnosti reprezentují pouze rozhraní síťové karty. Umožňuje nám také dynamickou stavbu virtuální sítě za použití výhradně virtuálních přepínačů, aby vznikla možnost konfigurace komunikace mezi jednotlivými virtuálními stroji. Hypervisor taktéž umožňuje komunikaci přes jeho fyzické síťové rozhraní a to tak, že umožní přidání fyzické síťové karty hypervisoru do logické infrastruktury hypervisoru. Tím je následně umožněna komunikace hypervisoru s virtuálními stroji a zajištěna komunikace z virtuálních strojů do externí sítě.



Obrázek 1.2: *Virtualizace pomocí hypervisoru*

Jedním z klíčových přínosů virtualizace sítí je vytvoření virtuálních přepínačů. Virtuální přepínač spojuje virtuální síťové karty do fyzické síťové karty serveru a spojuje virtuální síťové karty pro umožnění místní komunikace mezi virtuálními stroji. Velmi zajímavé je to, že díky virtuálnímu přepínači již nejsme závislí na fyzické síti. Zde se již klade důraz na velikost a rychlost operační paměti, která zajišťuje komunikaci virtuálních stojů a zároveň nám minimalizuje náklady na režie síťové infrastruktury. Tato úspora fyzické infrastruktury je proto, že virtuální stroje nemusí kvůli komunikaci mezi sebou posílat data přes fyzickou síťovou kartu, ale používá k tomu virtuálních síťových karet.

Nejdůležitějším prvek je přidání nových druhů přepínačů. Ty dostaly název distribuované virtuální přepínače. Umožňují nám přemostění mezi servery tak, že nechávají serverovou architekturu transparentní. Tím pádem virtuální přepínač na jednom serveru se může spojit s virtuálním přepínačem na serveru druhém (obrázek 1.3). Díky tomuto řešení se přesuny virtuálních strojů a jejich síťových rozhraní velmi zjednodušily, protože se mohou připojit k distribuovanému virtuálnímu přepínači na jiném serveru a transparentně se připojit do takové virtuální sítě.



Obrázek 1.3: *Spojení virtuálních síťových karet pomocí OpenvSwitch*

Jedním z nejdůležitějších projektů v tomto prostoru je technologie Open vSwitch. Dřívější implementace distribuovaných virtuálních přepínačů byla uzavřená pro operace s vlastními soubory hypervisorů. Ale v dnešních cloudových prostředích je nutná podpora heterogenních prostředí ve kterých tyto hypervisory mohou společně existovat. Open vSwitch je vícevrstvový virtuální přepínač, který je zdarma k použití pod licencí Apache 2.0. Skládá se z démona pro přepínač a společníka v podobě kernel modulu, který řídí samotné přepínání. Více k problematice [5].

2 Virtuální ethernetové přepínače

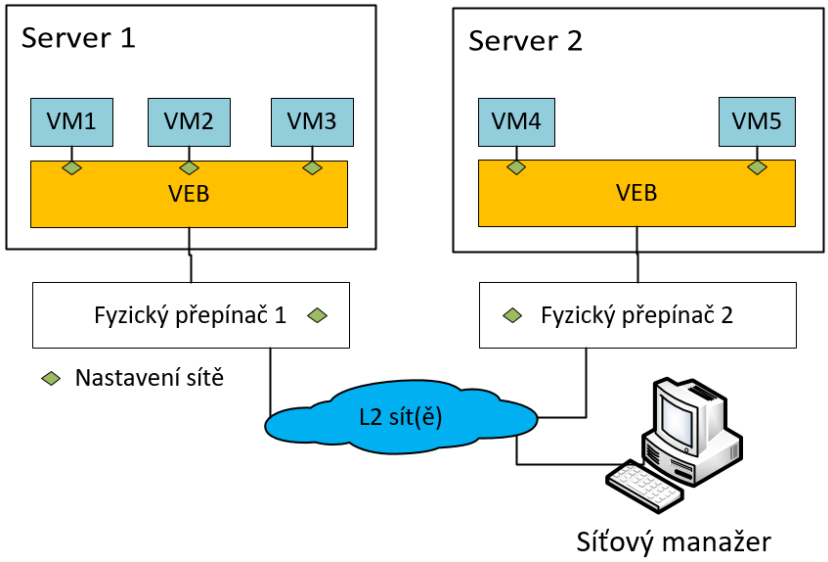
V dnešní době se ke správě a údržbě portů na každém přepínači počítačových sítí používají elementární nástroje. Do nastavení sítě spadají identifikátory VLAN, prvky pro ovládání portů, kontrola toku dat na daném portu a bezpečnostní pravidla pro kontrolu toku dat. Složitost těchto sítí se zvyšuje díky stále větší virtualizaci serverů, protože jeden fyzický přepínač spojuje vícero virtuálních strojů, kde každý virtuální stroj má síťové nastavení, které se udržuje právě na fyzickém přepínači. Virtuální stroje ale mohou používat virtuální přepínače, které taktéž obsahují informace o nastavení sítě. To platí hlavně u migrace virtuálního stroje, kdy síťové nastavení musí být při migraci přenesen přímo s virtuálním strojem.

2.1 Virtual Ethernet Bridging

Tato technologie není úplně nová, ale je zde již pár desetiletí (například zVM, PowerVM). Virtuální přepínání nám poskytuje efektivní komunikaci mezi více virtuálními stroji. Ve většině dnešních síťových implementací pro hypervisory (Virtual Switch, vSwitch, atd…) je již funkce migrace síťového nastavení obsažena. Nicméně, jedna z největších výzev spojená s dnešními přístupy k virtualizaci je automatizace přiřazení externího nastavení sítě danému virtuálnímu stroji a migrace tohoto nastavení během samotné migrace virtuálního stroje.

2.1.1 Homogenní nastavení sítě

V tomto případě mají všechny virtuální stroje stejné nastavení sítě. Typické použití tohoto modelu je například v situaci, kdy každý virtuální stroj používá stejný druh aplikace (nájemníka). Například je to, když všechny virtuální stroje používají stejnou aplikaci pro zobrazování webových stránek (APACHE).



Obrázek 2.1: *Homogenní nastavení sítě*

Obrázek 2.1 znázorňuje použití tohoto způsobu. Zde je vidět, jak se síťový manager používá ke konfiguraci stejného nastavení sítě na každém portu používaném servery pro přístup k přepínači. Všechny virtuální stroje jsou přiřazeny do stejného nastavení sítě ve fyzickém přepínači. V tomto případě, kdy dojde k migraci ze serveru 1 na server 2, nemusíme migrovat samotné nastavení sítě, protože je již toto nastavení sítě obsaženo i na fyzickém přepínači 2.

Tento příklad lze použít pro velké společnosti, kde mají velké množství racků a všechny servery používají stejné aplikace. Pokud by některý z těchto serverů byl příliš vytížen, jeden nebo více virtuálních strojů by se automaticky přesunul z jednoho serveru na druhý a to včetně jeho samotné síťové konfigurace, aby nedošlo k nedostupnosti virtuálního stroje po samotné migraci.

Bohužel, v tomto případě je hlavní omezení dané typem použité aplikace/nájemníka. Například, když bude virtuální stroj s web serverem přetížený a virtuální stroj s mail serverem nevyužitý, není možné převést virtuální stroj z webserveru na mail server, protože každá aplikace má jiné síťové nastavení.

2.1.2 Migrace virtuální stroje v heterogenní síti.

V tomto případě pracujeme s virtuálními stroji, které hostují různé typy nájemníků/aplikací. Například na některých virtuálních strojích běží aplikace k poskytování webových stránek a na jiných virtuálních strojích běží zase aplikace pro emailové servery. Nastavení sítě je přiřazeno každé aplikaci/nájemníkovi a toto nastavení je uloženo v obou VEB rozhraních na hypervisorech. Když virtuální stroj hostující určitou aplikaci/nájemníka začne svou migraci na jiný virtuální stroj, tak přiřazené síťové nastavení se musí taktéž migrovat.

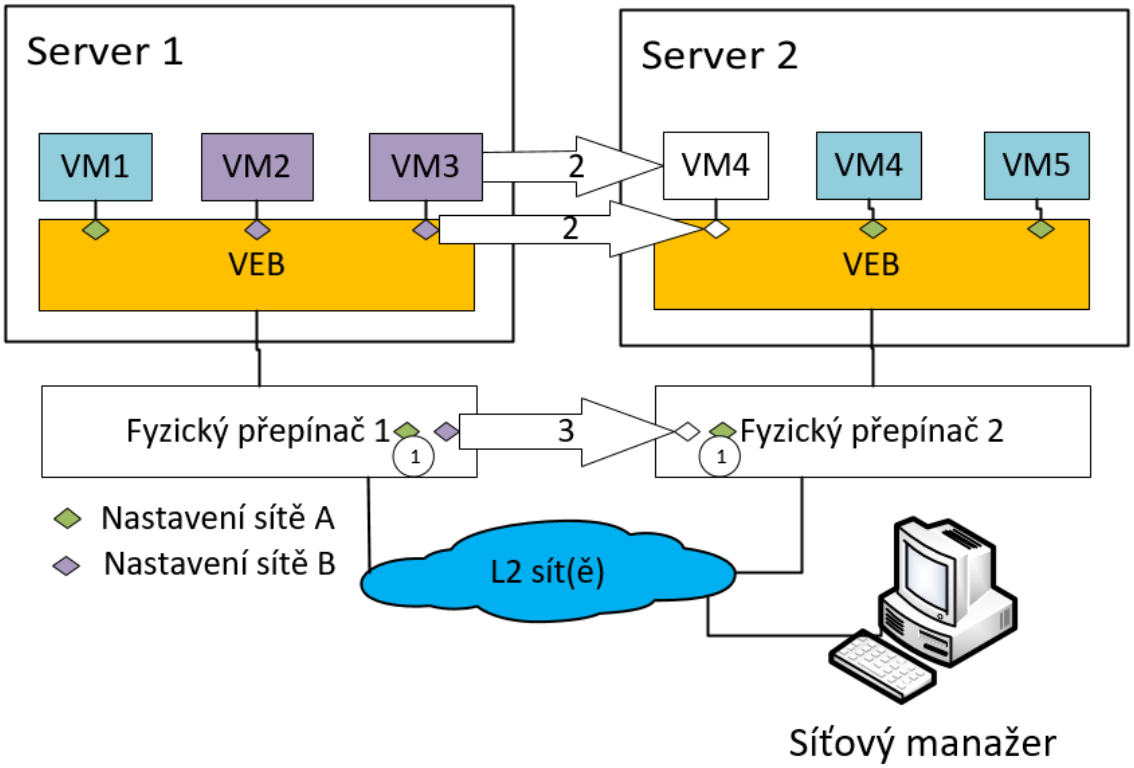
Dnes jsou dva způsoby, jak vyřešit tento požadavek. První způsob je použití MAC adresy k identifikaci typů aplikací/nájemníků a automatické přiřazení síťového nastavení pro danou MAC adresu po samotné migraci. Druhý způsob je manuální konfigurace priority síťového nastavení, kdy může být buď virtuální stroj migrován na jiný server, nebo dojde ke spuštění úplně nového virtuálního stroje na stávajícím serveru (což není automatizováno).

V tomto případě se použije síťový manager k vytvoření nastavení, které je unikátní pro každou aplikaci/nájemce. Hypervisor manager se používá k ovládání (vytvoření, startování, vypínání a odstranění) virtuálního stroje. Existují dva možné způsoby, jak šířit typ aplikace/nájemce, VLAN identifikátor a MAC adresu ve virtuálním stroji k síťovému manageru.

2.1.3 Manuální přiřazení

Když je MAC adresa přiřazena virtuálnímu stroji, společně s typem aplikace/nájemce, jsou tyto informace poslány ručně do síťového manageru. Poté síťový manager distribuuje tyto informace do přístupových přepínačů. Jakmile virtuální stroj začne komunikovat s přístupovým přepínačem, přepínač automaticky aplikuje síťové nastavení přiřazené na MAC adresu daného virtuálního stroje.

Na obrázku 2.2 vidíme příklad migrace virtuálního stroje pro tento způsob. V kroku 1 se přiřadí nastavení sítě virtuálnímu stroji, který používá tento způsob. V druhém kroku samotný hypervisor provede migraci virtuálního stroje, která obsahuje nastavení síťové migrace pro virtuální přepínač hypervisoru. V kroku 3, jakmile virtuální stroj začne používat svoji MAC adresu na Serveru 2, přepínač automaticky použije síťové nastavení přiřazené k této MAC adrese (původně na Serveru 1).



Obrázek 2.2: *Migrace virtuálního stroje pro heterogenní síťové nastavení*

2.1.4 Automatické přiřazení API

K automatizaci přiřazování a migraci síťového nastavení virtuálních stanic a jejich virtuálních rozhraní se používá API. Toto rozhraní je umístěno mezi hypervisorem a síťovým managerem. Používá se k vyjednání typu aplikace/nájemníka, MAC adresy a VLAN identifikátoru přiřazeného ke každé MAC adrese dané virtuální stanice. Stručně řečeno, bez API rozhraní nám hrozí, že přepínač nebude schopný rozlišit mezi reinkarnovanými a migrovanými MAC adresami. Tento přístup vyžaduje implementaci API pro každý hypervisor.

Reinkarnovaná MAC adresa: Je MAC adresa, která byla již dříve používána teď již odstraněným virtuálním strojem a nyní je používána jiným virtuálním strojem, který může mít naprosto odlišné síťové nastavení, oproti předchozímu virtuálními stroji, který tuto MAC adresu používal. Migrovaná MAC adresa: Je MAC adresa, která je přiřazena k virtuálnímu stroji, který

byl migrován mezi dvěma fyzickými servery v dané síťové struktuře a zůstává mu stejné síťové nastavení i po migraci.

Navíc, po přepínačích je požadována schopnost:

- Uchovat síťové nastavení pro všechny typy aplikací/nájemců – což vede ke spotřebě paměti přepínače
- Umožnit virtuálnímu stroji komunikaci, mezitím co přepínač načítá nastavení sítě po zahájení komunikace virtuálního stroje s přednastavenou MAC adresou, což vytvoří okno pro řízení přístupu (např. kde má virtuální stroj přístup k síti, bez toho, aby přepínač aplikoval přístupová/přenosová nastavení, která jsou přiřazena k dané aplikaci/nájemci daného virtuálního stroje.
- Nebo zakázaní komunikace virtuálního stroje, mezitím co přepínač získává nastavení sítě pro danou aplikaci/nájemce, což vede ke ztrátě paketů a může být velmi problematické při přenosu dat.

2.2 Otevřený přístup pro síťové nastavení heterogenní aplikace/nájemce

V tomto případě virtuální stroj hostuje odlišné typy aplikací/nájemců. To znamená, že všechny přepínače pro fyzický přístup jsou nastaveny jako porty trunku a VEB hypervisoru je samostatně odpovědný za nastavení nezbytného řízení přístupu a přenosu dat.

Problém s tímto přístupem je, že všechny fyzické servery musí být ve stejné bezpečnostní doméně, který má podobné omezení přesunu virtuálního stroje, jako homogenní nastavení sítě nájemce/služby. Například fyzický server nemůže být řízen pomocí aplikace/nájemce A ve stejné druhé vrstvě sítě LAN, stejně jako fyzický server řízený aplikací/nájemcem B.

2.3 Shrnutí používání VEB

Dnes již existuje několik způsobů pro migraci síťového nastavení přiřazeným k virtuálnímu síťovému rozhraní virtuálního stroje. Bohužel, téměř každý způsob v sobě nese omezení, které mohou způsobit bezpečnostní riziko, nebo omezení při migraci.

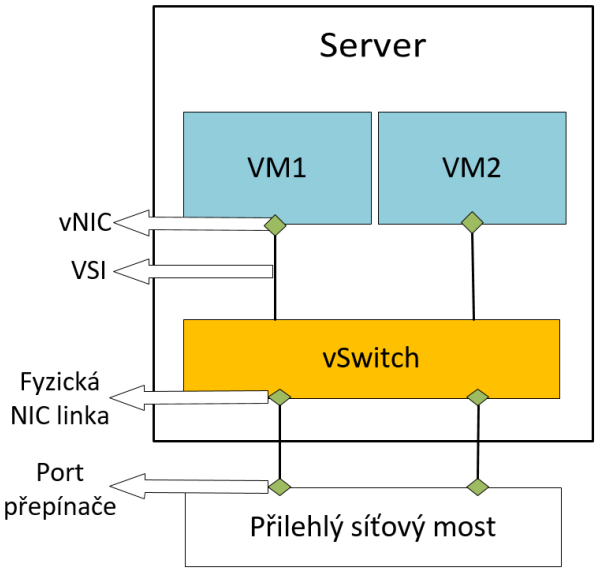
3 Virtual Ethernet Bridging

Abychom zautomatizovali vytvoření a migraci síťového nastavení, potřebujeme k tomu plošný kontrolní protokol, který přiřazuje a odebírá virtuální stroje a jejich rozhraní podle nastavení na nejbližším ovladači fyzického síťového rozhraní, které používá daný virtuální stroj. V této části se budu zabývat stručným přehledem možných řešení, které by nám umožnili toto sjednocování. Více k problematice [6]

3.1 Definování stavů sítě a její možnosti

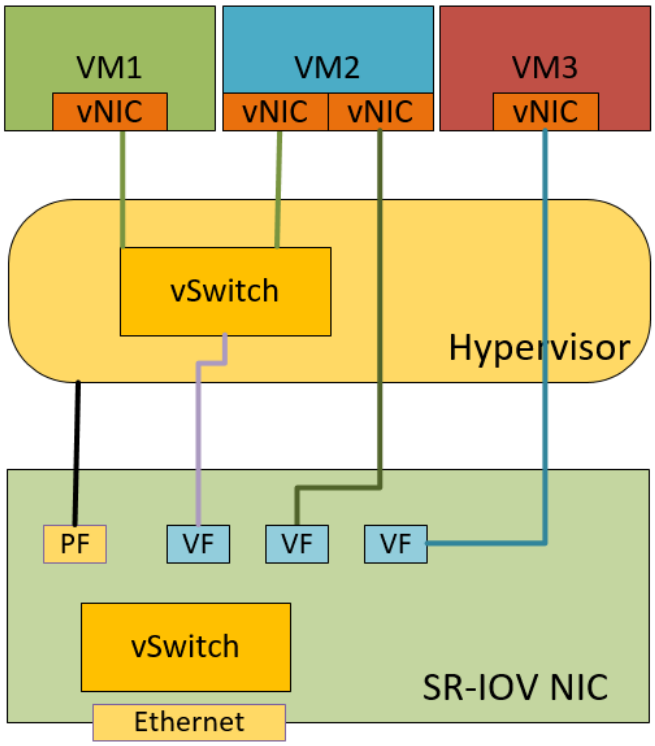
Na obrázku 3.1 je ukázáno spojení serveru na přilehlý síťový most (přepínač). V tomto případě má server dva virtuální stroje, kde každý virtuální stroj má dvě virtuální síťová rozhraní. Každá z vNIC je připojena k vSwitch hypervisoru pomocí VSI. VSwitch může být implementován jako VEB, nebo jako VEPA. Jeden běžící hypervisor může obsahovat jeden a více vSwitchů.

VEB provádí směrování portu z prvního VSI na druhé VSI stejně dobře, jako by se směrovalo z VSI do přilehlého síťového mostu. Použitý vSwitch na obrázku 3.1 lze implementovat ve fyzické NIC.



Obrázek 3.1: *Části virtuální stanice*

Ukázka tohoto způsobu je na obrázku 3.2, kde můžeme vidět Single Root Input/Output Virtualization NIC (SR-IOV NIC), která podporuje jednu fyzickou funkci (PF) a několik virtuálních funkcí (VF). Každá z těchto funkcí je připojena do vSwitche SR-IOV NIC přes VSI. Stejně jako hypervisor založený na vSwitchi, taktéž vSwitch v SR-IOV NIC může být implementován jako VEB, nebo jako VEPA. Jeden SR-IOV NIC může obsahovat několik takových vSwitchů.



Obrázek 3.2: *vSwitch založený na NIC*

4 Linuxové kontejnery

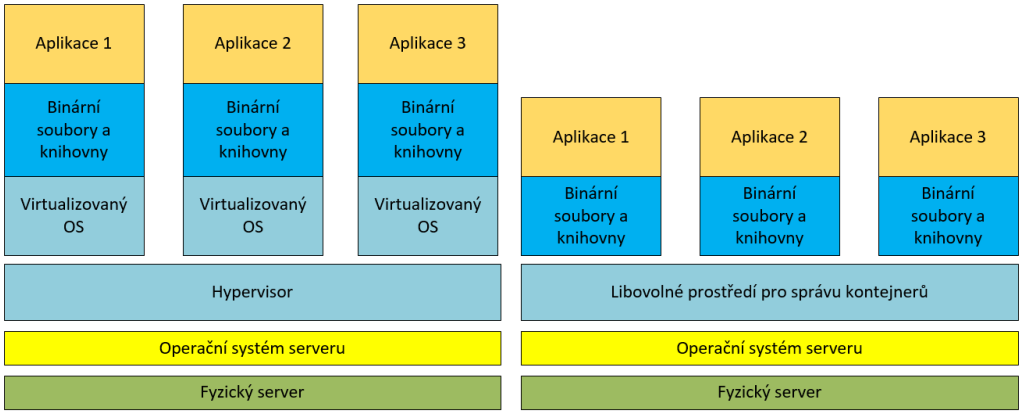
Linuxový kontejner je soubor procesů, které jsou isolované od zbytku systému. Kontejnery běží na samostatném obrazu, který poskytuje všechny soubory potřebné k podpoře kontejnerových procesů. Tím, že kontejneru poskytujeme obraz, který obsahuje všechny potřeby aplikace, je tento kontejner konzistentní při přesunu například z vývoje do testování a z testování do běžného použití.

Představme si, že například vyvíjíme nějakou aplikaci. Doma můžeme pracovat na PC, které má specifickou konfiguraci. Ostatní vývojáři ale můžou pracovat na počítačích s odlišnou konfigurací. Aplikace, kterou vyvíjíme, závisí na této konfiguraci a k chodu potřebuje specifické soubory. Firma, pro kterou tuto aplikaci vyvíjíme, má vlastní testovací a produkční prostředí, které je standardizované s jejich vlastním balíčkem podporných souborů. Při vývoji se chceme co nejblíže přiblížit prostředí, do kterého bude aplikace nasazena. Nechceme přitom vytvářet složitý server, který je stejný, jako ten firemní.

Tohoto lze jednoduše dosáhnout právě pomocí kontejnerů. Díky kontejnerům, jsme schopni přesouvat jednotlivé aplikace, nebo služby na různé systémy. To vše díky tomu, že v kontejnerech je obsaženo celé prostředí, potřebné pro běh naší aplikace. Obsahuje nastavení a potřebné soubory k jejímu spuštění. Díky tomu lze naši aplikaci používat jak na našem pracovním počítači, tak i na firemních serverech bez problémů s jednotlivými konfiguracemi systémů.

4.1 Virtualizace a kontejnery

Virtualizace nám umožňuje běh více operačních systémů v jeden čas na jediném systému. Kontejnery sdílejí stejné jádro operačního systému a izolují procesy aplikací od zbytku systému.



Obrázek 4.1: *Rozdíl mezi virtualizací a kontejnery*

U virtualizace máme vícero operačních systémů běžících na hypervisoru, který zajišťuje chod virtualizací, ale není tak odlehčený (bere více systémových prostředků), než kontejner. Pokud máme omezené zdroje s omezenou kapacitou, potřebujeme odlehčené aplikace, které mohou být hustě nasazeny. Linuxové kontejnery běží z jednotného operačního systému a tento systém sdílí skrze všechny kontejnery, takže aplikace a služby zůstávají odlehčené a fungují svižně při paralelním spuštěním.

4.2 Historie kontejnerů

Prvotní nápad kontejnerů vznikl již v roce 2000 a byla jím technologie nazývaná FreeBSD Jail. Technologie, která nám umožnila rozdělit FreeBSD systém do více podsystémů. Tato vězení byla vytvořena jako bezpečné prostředí, které mohl systémový administrátor sdílet s více uživateli uvnitř nebo venku organizace. V těchto vězeních bylo záměrem, aby se procesy vytvářely v modifikovaném chroot prostředí, kde je přístup do systému souborů, sítě a uživatelských účtů omezen. Díky tomu tyto procesy nemůžou uniknout, nebo kompromitovat daný systém.

4.3 LXC a LXD

LXC a LXD jsou dvě důležité zkratky, které bychom měli znát, pokud nás zajímají kontejnery. Bohužel, jsou to zkratky, které se od sebe těžko rozeznávají. Odpovídají stejným platformám, které byly vytvořeny z velké části stejnou společností. Na technické úrovni jsou tyto zkratky úzce spojené.

4.3.1 LXC

LXC je zkratka pro Linux containers, je řešení pro virtualizační software na úrovni operačního systému v rámci Linuxové kernelu (jádra). Na rozdíl od klasických hypervisoru (VMware, KVM, Hyper-V), LXC nám umožňuje spouštět jednotlivé aplikace ve virtuálním prostředí. Nicméně pokud budeme chtít, je možné virtualizovat celý operační systém uvnitř LXC kontejneru. Hlavními výhodami LXC je kontrola nad virtuálním prostředím za použití namespaces nástrojů z operačního systému hosta (serveru), dále pak nižší režie než klasický hypervisor a větší přenositelnost individuálních aplikací mezi jednotlivými kontejnery. LXC stojí i za řadou jiných nástrojů jako Docker a CoreOS. Každopádně technologie LXC je původní koncepce kontejnerů, která vznikla před několika lety a její hodnoty/principy zůstávají základem pro vývoj dalších kontejnerových systémů.

4.3.2 LXD

Zjednodušeně bychom mohli LXD definovat jako rozšíření pro LXC. Jelikož důležitost LXC pro Docker a CoreOS přestala být klíčová, stalo se LXD hlavním pilířem úspěchu LXC. Z technického hlediska můžeme říct, že LXD se dá popsat jako zbytek API, který se připojuje na softwarovou knihovnu LXC nazvanou liblxc. LXD rozšíření, které je napsané v Go, vytváří systémového démona který je dostupný pomocí lokálního Unix socketu, nebo pomocí sítě s HTTPS.

4.3.3 Hlavní výhody LXD

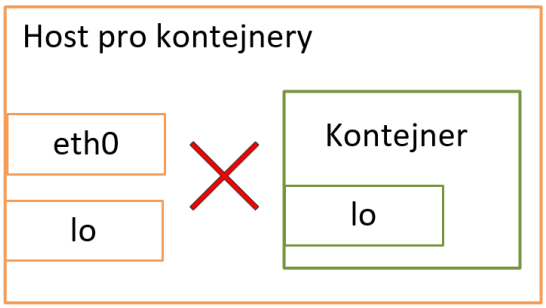
Na hostu může běžet spousta LXC kontejnerů, které používají pouze jednoho systémového démona, což usnadňuje řízení a snižuje náklady na režii. Při použití pouze LXC, musíme vždy tyto procesy rozdělit na jednotlivé kontejnery. LXD démon může získat výhody jako server, které dělají kontejnery více zabezpečenými. Při použití holého LXC je zabezpečení problematičtější. LXD démon se zabývá sítí, datovými úložišti a zároveň má uživatel možnost ovládat tyto prvky z LXD CLI rozhraní. To zjednodušuje proces sdílení zdrojů s jednotlivými kontejnery. LXD nabízí rozšířené vlastnosti, které LXC nemá, jako jsou migrace aktivního kontejneru a možnost zálohovat běžící kontejner.

4.4 LXC síťové konfigurace

V této kapitole popíšu různá nastavení virtuálních síťových rozhraní. Toto je velmi důležitá část této práce, jelikož zde popíšu veškeré způsoby komunikace mezi různými virtuálními stanice. Více k problematice [7]**Chyba! Nenalezen zdroj odkazů.**

4.4.1 Empty

Vytváří pouze rozhrání loopback, neboli smyčka. Rozhraní se používá pouze pro testování, kdy rozhraní dostane stejné informace, jaké odeslal virtuální stroj na toto rozhraní. Na kontejner s tímto typem sítě se nelze síťově připojit. Spravovat jej lze pomocí konzole implementované ve správci kontejnerů.



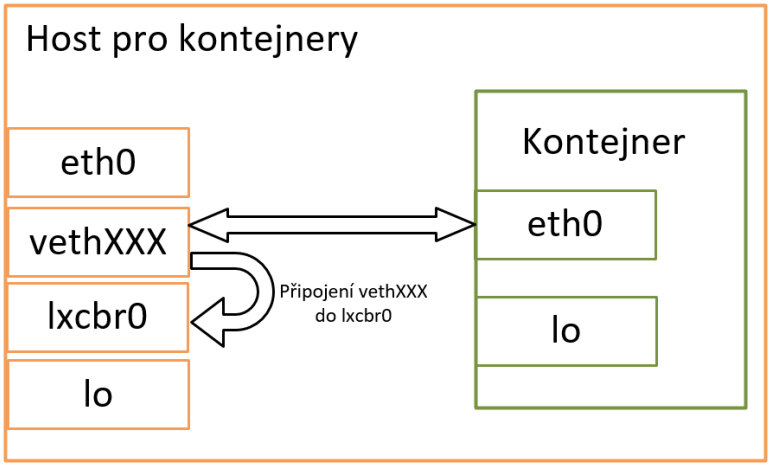
Obrázek 4.2: *Ukázka empty sítě*

4.4.2 Veth

Je síťové zařízení vytvořeno s jednostranným přiřazením ke kontejneru a na druhé straně je přiřazeno do mostu, který je specifikován v konfiguračním souboru pod příkazem lxc.network.link. Pokud není žádný most specifikován, vytvoří se pouze veth zařízení v kontejneru, ale nebude přiřazeno žádnému mostu. V opačném případě je potřeba nakonfigurovat samotný most předem v daném systému, protože jinak lxc nebude zvládat konfiguraci mimo daný kontejner. Ve výchozím nastavení se jméno pro veth vybírá podle nastavení, které řídí přímo lxc. Jeho tvar je potom například veth7fZ56sawe47. Pokud chceme

specifikovat toto jméno, abychom se v tom trochu vyznali, je potřeba říct lxc, jaký název má na toto veth rozhraní přiřadit. Toto nastavení se provádí přímo na kontejneru v souboru config, který slouží k nastavení virtuálního síťového zařízení. Příkaz, který se přidá do configuračního souboru je:

```
lxc.network.veth.pair = nazevrozhrani
```



Obrázek 4.3: *Ukázka veth kontejneru*

4.4.3 **Macvlan**

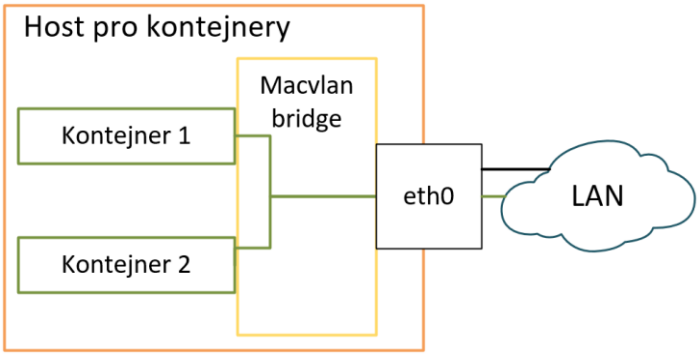
Toto rozhraní se spojuje s rozhraním, které se specifikuje pomocí příkazu

```
lxc.network.link = nazevrozhrani
```

a je přiřazeno danému kontejneru. Příkaz

```
lxc.network.macvlan.mode
```

pak udává způsob, jakým bude macvlan komunikovat mezi dvěma odlišnými macvlany na stejném nadřazeném zařízení.



Obrázek 4.4: *Ukázka macvlan kontejneru*

4.4.4 **Vlan**

Vlan rozhraní je spojeno s rozhraním specifikovaném pomocí příkazu:

```
lxc.network.link
```

a přiřazeno kontejneru. Jméno identifikátoru se dá nastavit v konfiguračním souboru sítě pomocí příkazu:

```
lxc.network.vlan.id = nazevvlan
```

U tohoto nastavení se předpokládá, že bude použito síťového přepínače s technologií reflective relay.

4.4.5 **Phys**

Toto nastavení by se mělo chovat jako bychom použili přímo fyzickou síťovou kartu našeho fyzického hosta. U tohoto nastavení mě bude hlavně zajímat, jestli nedojde k přerušení komunikace mezi jinými kontejnery, které by aktuálně používaly dané síťové rozhraní, jenž jsme přiřadili virtuálnímu kontejneru s phys nastavením jeho virtuální sítě. Již existující zařízení specifikováno pomocí parametru,

```
lxc.network.link
```

který se přiřadí kontejneru. V podstatě jde o využití fyzické síťové karty přímo od fyzického počítače.

5 Praktická část

5.1 Instalace serveru:

Pro instalaci serveru, na kterém se bude provádět měření, jsem zvolil klasický PC s dvoujádrovým procesorem, plotnovým diskem a 8 GB operační paměti. Jako operační systém jsem si vybral distribuci se síťovou instalací Linux Ubuntu 16.04. Po nainstalování operačního systému jsem provedl instalaci LXC, ve kterém bych chtěl kontejnery spravovat. Více k problematice [7]**Chyba! Nenalezen zdroj odkazů.**

Po instalaci LXC se nám vytvoří virtuální rozhraní s názvem lxcbr0, které potom slouží jako výchozí brána pro samotné kontejnery s nastavením veth. Pro ostatní kontejnery můžeme vytvořit vlastní rozhraní, na který budou komunikovat.

```
lxcbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
 link/ether 00:16:3e:00:00:00 brd ff:ff:ff:ff:ff:ff
 inet 10.0.3.1/24 scope global lxcbr0
```

Obrázek 5.1: *Ukázka lxcbr0 rozhraní*

Jelikož chceme mít v síti IPv6 adresy, je potřeba stáhnout program radvd a nakonfigurovat ho, aby rozděloval IPv6 adresy na rozhraní, které potřebujeme spojit pomocí globálních IPv6 adres. Adresy lze samozřejmě nakonfigurovat i ručně v síťovém nastavení kontejneru. Pro vytvoření kontejneru slouží jednoduchý příkaz:

```
lxc-create –n nazevkontejneru
```

Po použití parametru –t se nám stáhne celá šablona z repositáře kontejnerů. Prvotní stažení trvá trošku déle, ale každý další kontejner s touto šablonou je vytvořen během pár sekund.

```
ztr@ubuntu:~$ sudo lxc-create -n kontejnerempty  -t ubuntu
[sudo] password for ztr:
Checking cache download in /var/cache/lxc/xenial/rootfs-amd64 ...
Copy /var/cache/lxc/xenial/rootfs-amd64 to /var/lib/lxc/kontejnerempty1/rootfs .
..
Copying rootfs to /var/lib/lxc/kontejnerempty1/rootfs ...
Generating locales (this might take a while)...
  en_US.UTF-8... done
Generation complete.
Creating SSH2 RSA key; this may take some time ...
2048 SHA256:CLrR2ECcmUZzEp0Gv9UQAg292I5b2sjXkf/a7mzrKXo root@ubuntu (RSA)
Creating SSH2 DSA key; this may take some time ...
1024 SHA256:x0InIkORiIyz+Wwf2EfYWBS6jl0Rf7CciBJPr0LeHUA root@ubuntu (DSA)
Creating SSH2 ECDSA key; this may take some time ...
256 SHA256:n9rASpMmWO8lRAyd5tKO+rXNW2Rr3KuP/PUsODbjSyQ root@ubuntu (ECDSA)
Creating SSH2 ED25519 key; this may take some time ...
256 SHA256:qYsuM/hJtYxi0COlOHWv6HNKEleqM8dRsrsmsuxDSVA root@ubuntu (ED25519)
invoke-rc.d: could not determine current runlevel
invoke-rc.d: policy-rc.d denied execution of start.

Current default time zone: 'Etc/UTC'
Local time is now:      Mon Apr 23 12:49:01 UTC 2018.
Universal Time is now:  Mon Apr 23 12:49:01 UTC 2018.


##
# The default user is 'ubuntu' with password 'ubuntu'!
# Use the 'sudo' command to run tasks as root in the container.
##
```

Obrázek 5.2: *Úspěšná instalace kontejneru*

S vytvořením každého kontejneru se vytvoří i virtuální síťové rozhraní pro daný kontejner. Ten můžeme vidět zde:

```
ztr@ubuntu:~$ ifconfig
kontejnerveth Link encap:Ethernet  HWaddr fe:02:f3:c6:fd:e4
          inet6 addr: fe80::fc02:f3ff:fec6:fde4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:30 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:508 (508.0 B)  TX bytes:4181 (4.1 KB)
```

Obrázek 5.3*: Výpis virtuálního rozhraní na hostu pro kontejner veth*

Jelikož by název výchozího rozhraní vypadal nějak takhle veth7Fetgx, upravil jsem jeho název konfiguračním souboru kontejneru na kontejnerveth. Toto lze provést přidáním příkazu do konfiguračního souboru sítě kontejneru. Příkaz pro pojmenování virtuálního rozhraní je:

```
lxc.network.veth.pair = nazevrozhrani
```

5.2 Praktické nastavení různých typů sítě

5.3 Empty

Vytvoříme si nový kontejner s názvem kontejnerempty s parametrem –t ubuntu, který zajistí nahrání šablony s obrazem OS Ubuntu.

```
Sudo lxc-create -n kontejnerempty -t ubuntu
```

Nyní můžeme nahlédnout, co se uvnitř tohoto kontejneru skrývá pomocí příkazu. Na obrázku vidíme, že uvnitř nového kontejneru se nachází soubor config, který pro nás bude velmi užitečný, jelikož v sobě skrývá nastavení sítě.

```
ztr@ubuntu:~$ sudo ls -l /var/lib/lxc/kontejnerempty
total 8
-rw-r--r--  1 root root  766 Apr 18 10:18 config
drwxr-xr-x 21 root root 4096 Apr 18 10:22 rootfs
```

Obrázek 5.4: *Obsah kontejneru*

Ve výchozím nastavení je síťová konfigurace nastavena na veth.

```
# Network configuration
lxc.network.type = veth
lxc.network.link = lxcbr0
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:39:86:39
```

Obrázek 5.5: *Nastavení síťového rozhraní kontejneru*

Tu tedy změníme na empty.

```
# Network configuration
lxc.network.type = empty
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:39:86:39
```

Obrázek 5.6: *Nastavení sítě pro empty stav*

Nyní máme konfiguraci sítě hotovou a můžeme nastartovat kontejnerempty.

```
ztr@ubuntu:~$ sudo lxc-ls --fancy
NAME           STATE   AUTOSTART GROUPS IPV4 IPV6
kontejnerempty STOPPED 0         -      -    -
ztr@ubuntu:~$ sudo lxc-start -n kontejnerempty
ztr@ubuntu:~$ sudo lxc-ls --fancy
NAME           STATE   AUTOSTART GROUPS IPV4 IPV6
kontejnerempty RUNNING 0         -      -    -
```

Obrázek 5.7: *Start kontejneru s empty nastavením 1*

Z obrázku je patrné, že kontejner běží, ale neobdrželi jsme žádnou IP adresu. To je samozřejmě správně, protože jak jsem zmínil výše, tak druh sítě empty podporuje pouze rozhraní loopback.

Toto nové rozhraní nebude nikdy vidět na straně hosta. Jedná se o lokální virtuální rozhraní kontejneru. Přesně tak je to uvedeno i v dokumentaci LXC, takže si pojďme ověřit, jestli se nám opravdu vytvořilo pouze rozhraní loopback.

```
ztr@ubuntu:~$ sudo ip link list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
 group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
mode DEFAULT group default qlen 1000
    link/ether 00:1e:8c:7a:db:59 brd ff:ff:ff:ff:ff:ff
```

Obrázek 5.8: *Výpis rozhraní na hostu*

Z výpisu je patrné, že se nám nevytvořilo žádné virtuální rozhraní pro náš kontejner kontejnerempty. Nyní to můžeme ověřit i na straně kontejneru. Přihlásíme se do konsole našeho kontejnerempty pomocí příkazu:

```
sudo lxc-console –n kontejnerempty
```

A následně pomocí příkazu:

```
sudo ip a
```

vypíšeme list našich síťových rozhraní.

```
ubuntu@kontejnerempty:~$ sudo ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
```

Obrázek 5.9: *Kontrola rozhraní na kontejnerempty*

Zde můžeme vidět, že je na kontejneru vytvořené opravdu jenom interface lo, čili loopback.

Rozhraní loopback je virtuální zařízení, které je implementovánu přímo v Kernelu a host ho využívá ke komunikaci sám se sebou. Zařízení (rozhraní) má přiřazený speciální nesměrovatelný adresový blok IP adres 127.0.0.0/8, a pro IPv6 je to ::1/128, takže IP adresa 127.0.0.1 skrze masku 127.255.255.254 může být použita ke zpětné komunikaci s hostem. Linuxové, ale i Windows distribuce překládají tuto IP adresu jako localhost. Toto rozhraní slouží pro diagnostiku, ladění, testování, nebo tam, kde prostě nechceme, aby naše služby byly zobrazeny v žádné síti. Na obrázku 5.10 můžeme vidět směrovací tabulku našeho kontejneru s empty nastavením.

```
ubuntu@kontejnerempty:~$ sudo ip route show table local
broadcast 127.0.0.0 dev lo  proto kernel  scope link  src 127.0.0.1
local 127.0.0.0/8 dev lo  proto kernel  scope host  src 127.0.0.1
local 127.0.0.1 dev lo  proto kernel  scope host  src 127.0.0.1
broadcast 127.255.255.255 dev lo  proto kernel  scope link  src 127.0.0.1
```

Obrázek 5.10: *Routovací tabulka empty nastavení*

5.4 Veth

K prozkoumání druhu sítě Veth si vytvoříme nový kontejner se stejnou šablonou, ale s jiným nastavením sítě. Pomocí příkazu

```
sudo lxc-create –n kontejnerveth –t ubuntu
```

opět vytvoříme kontejner s názvem kontejnerveth. Jelikož jsme tuto šablonu stáhli v předchozí konfiguraci rozhraní empty a lxc si ji uložilo, tak nám vytvoření nového kontejneru trvalo jen pár sekund. Zkontrolujeme tedy, jestli se kontejner vytvořil.

```
ztr@ubuntu:~$ sudo lxc-ls --fancy
NAME           STATE   AUTOSTART GROUPS IPV4 IPV6
kontejnerempty RUNNING 0         -      -    -
kontejnerveth  STOPPED 0         -      -    -
```

Obrázek 5.11: *Vytvořený kontejner pro veth rozhraní*

Pojem peer device můžeme chápat jako pár falešných ethernetových zařízení, které se chovají jako trubka nebo tunel. To znamená, že data poslaná přes jedno rozhraní vždy přijdou na rozhraní druhé. Můžeme tedy očekávat, že když spustíme kontejner s nastaveným typem sítě veth, vytvoří se nám jedno síťové rozhraní na hostu a jedno na kontejneru. Následně síťové rozhraní hosta bude přemostěno na virtuální most lxcbr0, který bude spojen s naším veth rozhraním v kontejneru.

Nyní pojďme nastavit síťově rozhraní do našeho kontejneru pomocí následné konfigurace souboru config. Nejprve zjistíme, jaké nastavení náš soubor config skrývá pomocí příkazu:

```
sudo grep network /var/lib/lxc/kontejnerveth/config
```

```
ztr@ubuntu:~$ sudo grep network /var/lib/lxc/kontejnerveth/config
lxc.network.type = veth
lxc.network.link = lxcbr0
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:7d:0a:a1
```

Obrázek 5.12: *Výpis souboru config pro kontejnerveth*

Jak si můžeme všimnout, tak výchozí síťové nastavení při použití příkazu:

```
Sudo lxc-create
```

je typu veth. Není tedy nutné nic nastavovat. Lze například změnit jméno rozhraní, ale k tomu se dostanu později.

Nyní spustíme kontejnerveth a zkontrolujeme stav kontejneru.

```
ztr@ubuntu:~$ sudo lxc-ls --fancy
NAME           STATE   AUTOSTART GROUPS IPV4            IPV6
kontejnerempty STOPPED 0         -      -               -
kontejnerveth  RUNNING 0         -      10.0.3.167      2001:db8:1:0:216:3eff:fe7d:aa1
```

Obrázek 5.13: *Spuštění kontejneru a získání IP adres*

Jak můžeme vidět, tak kontejner se spustil a hned si vzal IP adresu, kterou dostal skrze virtuální rozhraní lxcbr0. Je zjevné, že LXC již obsahuje DHCP server, který lze dále konfigurovat. IPv6 adresu získal kontejner z programu radvd, který jsme si spustili pro rozdělování IPv6 adres.

Postup získání IP adresy.

1. Na hostu se vytvořil pár virtuálních síťových rozhraní. Nově vzniklé kontejnery jsou, co se sítě týče, konfigurovány pomocí DHCP serveru (jedná se o dnsmasq démona), který naslouchá na IP adrese přiřazené na LXC síťový most. V našem případě jde o rozhraní lxcbr0. Můžeme to ověřit pomocí příkazu:

```
sudo netstat –ntpl | grep 53
```

IP adresa mostu slouží zároveň jako výchozí brána a taktéž jako její nameserver.

2. Hostova část páru daného veth zařízení je posílána skrze most, tak jak je nastaveno v daném kontejneru. V našem případě je to právě rozhraní lxcbr0.

3. Druhá část veth zařízení je potom přesunuta na kontejner, kde se jmenuje eth0 a je nakonfigurována ve jmenné části kontejneru.

4. Jakmile kontejnerový proces init nastartuje, přivede nám k životu rozhraní síťového zařízení v kontejneru a může začít komunikovat se světem.

Nyní tedy zkusíme, jestli je náš kontejner dostupný z jeho hosta. K tomu nám poslouží obyčejný příkaz PING.

```
ztr@ubuntu:~$ ping6 2001:db8:1:0:216:3eff:fe7d:aa1
PING 2001:db8:0:1:216:3eff:feed:cace(2001:db8:0:1:216:3eff:feed:cace) 56 data bytes
64 bytes from 2001:db8:1:0:216:3eff:fe7d:aa1: icmp_seq=1 ttl=64 time=0.056 ms
64 bytes from 2001:db8:1:0:216:3eff:fe7d:aa1: icmp_seq=2 ttl=64 time=0.087 ms
64 bytes from 2001:db8:1:0:216:3eff:fe7d:aa1: icmp_seq=3 ttl=64 time=0.089 ms
```

Obrázek 5.14: *Obrázek 2: Ping z hosta na kontejner veth*

Příkaz ping byl úspěšný a obdrželi jsme odezvu od našeho kontejneru.

Nyní se přihlásíme do konsole našeho kontejneru a podíváme se, jestli naše domněnky ohledně rozhraní byly správné. Pomocí následujícího příkazu se připojíme ke konsoli s údaji ubuntu/ubuntu,

```
lxc-console –n kontejnerveth
```

Dále pomocí příkazu

```
sudo ip
```

nahlédneme na naše síťová rozhraní.

```
ubuntu@kontejnerveth:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
12: eth0@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
 group default qlen 1000
    link/ether 00:16:3e:7d:0a:a1 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.3.167/24 brd 10.0.3.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 2001:db8:1:0:216:3eff:fe7d:aa1/64 scope global mngtmpaddr dynamic
       valid_lft 86397sec preferred_lft 14397sec
    inet6 fe80::216:3eff:fe7d:aa1/64 scope link
       valid_lft forever_preferred_lft forever
```

Obrázek 5.15: *Výpis IP adres a routy*

Veth pár na kontejneru je přesně takový, jaký jsme očekávali. Rozhraní eth0 má přiřazenou korektní adresu a jeho síť je nakonfigurována, aby používala bránu našeho lxcbr0 mostu.

Nyní se podíváme ještě na síťové rozhraní hosta.

```
13: vethLGK4X3@if12: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue maste
r lxcbr0 state UP group default qlen 1000
    link/ether fe:ec:8f:17:eb:41 brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::fcec:8fff:fe17:eb41/64 scope link
       valid_lft forever preferred_lft forever
```

Obrázek 5.16: *Vytvořený veth interface na straně hosta*

Je zjevné, že na straně hosta došlo k vytvoření veth síťového rozhraní. Nyní se podíváme, jestli došlo k přemostění rozhraní veth na náš lxcbr0.

Použijeme příkaz

```
sudo brctl show
```

```
ztr@ubuntu:~$ sudo brctl show
bridge name     bridge id               STP enabled     interfaces
lxcbr0          8000.00163e000000       no              vethLGK4X3
```

Obrázek 5.17: *Virtuální rozhraní připojená k lxcbr0*

Příklady použití veth:

- vytvoření virtuální sítě mezi kontejnery a jejich následné spojení pomocí mostu na různých sítích.
- Poskytnutí směrovaných linek pro kontejner, které slouží ke komunikaci s vnější sítí.

- emulace přemostěných sítí, sloužící například k lokálnímu testování složitějších nastavení sítě.
- lze testovat téměř jakoukoliv topologii.

5.5 Bridge

Poskytuje nám chování jednoduchého mostu mezi odlišnými MAC VLAN rozhraními na stejném portu. Rámce z jednoho rozhraní do druhého jsou doručeny přímo a neposílají se externě, což nám zaručuje zabezpečení pro vnitřní komunikaci mezi kontejnery. Nahlížíme na to jako na zjednodušený přepínač, který se nepotřebuje učit MAC adresy, jelikož je už zná.

Umožní nám vytvoření speciálního mostu (ne klasického jako lxcbr0), který je ovšem izolovaný od klasického mostu nadřazeného hosta (lxcbr0). Tím nám zajistí komunikaci mezi jednotlivými kontejnery bez nutnosti obtěžování hosta.

Tento most využívám pro komunikaci mezi kontejnery pro macvlan, V této práci se poté označuje jako lxcbr1. Tento most je použit pro kontejenermvlb1 a kontejenrmvlb2, kde zajišťuje jejich izolovanou komunikaci.

Abych ukázal, jak tento druh mostu funguje, vytvořím si 2 LXC kontejnery, které přemostíme pomocí ručně vytvořeného mostu za použití LXC síťového mostu, který nám poskytne MAC VLAN síť.

Nejprve si vytvořím na hostu nový síťový most, který pojmenujme třeba lxcbr1 a to příkazem

```
sudo brctl addbr lxcbr1
```

a následně tento most nastartujeme pomocí příkazu

```
sudo ip link set lxcbr1 up
```

```
ztr@ubuntu:~$ sudo brctl show
bridge name     bridge id               STP enabled     interfaces
lxcbr0          8000.00163e000000       no              vethLGK4X3
lxcbr1          8000.000000000000       no
```

Obrázek 5.18: *Výpis nově vytvořeného mostu*

Nyní si vytvoříme kontejnery a nakonfigurujeme je pro použití macvlan sítě v bridge macvlan stavu. Zároveň jim nastavíme IP adresy v souboru config (jelikož víme, že výchozí nastavení je veth stav), aby spolu mohli kontejnery komunikovat.

```
ztr@ubuntu:~$ sudo lxc-ls --fancy
NAME           STATE   AUTOSTART GROUPS IPV4          IPV6
kontejnerempty STOPPED 0         -      -             -
kontejnermvlb1 STOPPED 0         -      -             -
kontejnermvlb2 STOPPED 0         -      -             -
```

Obrázek 5.19: *Vytvořené kontejnery pro macvlan stav*

Zkontrolujeme si konfiguraci sítě na obou nově vytvořených kontejnerech.

```
ztr@ubuntu:~$ sudo grep network /var/lib/lxc/kontejnermvlb1/config
lxc.network.type = macvlan
lxc.network.macvlan.mode = bridge
lxc.network.link = lxcbr1
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:dd:d7:70
lxc.network.ipv4 = 10.0.5.3/24
lxc.network.ipv6 = 2001:db8:1:10::10/64
ztr@ubuntu:~$ sudo grep network /var/lib/lxc/kontejnermvlb2/config
lxc.network.type = macvlan
lxc.network.macvlan.mode = bridge
lxc.network.link = lxcbr1
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:b1:4b:ef
lxc.network.ipv4 = 10.0.5.4/24
lxc.network.ipv6 = 2001:db8:1:10::20/64
```

Obrázek 5.20: *Síťové nastavení kontejneru pro macvlan*

Konfigurace je nastavena a nyní spustíme oba kontejnery a zkontrolujeme jejich stav.

```
NAME           STATE   AUTOSTART GROUPS IPV4          IPV6
kontejnerempty STOPPED 0         -      -             -
kontejnermvlb1 RUNNING 0         -      10.0.5.3      2001:db8:1:10::10
kontejnermvlb2 RUNNING 0         -      10.0.5.4      2001:db8:1:10::20
kontejnerveth  RUNNING 0         -      10.0.3.167    2001:db8:1:0:216:3eff:fe7d:aa1
```

Obrázek 5.21: *Macvlan kontejnery s přiřazenými IP adresami*

Nyní prověříme komunikaci mezi macvlan rozhraními.

```
ubuntu@kontejnermvlb2:~$ ping6 2001:db8:1:10::10
PING 2001:db8:1:10::10(2001:db8:1:10::10) 56 data bytes
64 bytes from 2001:db8:1:10::10: icmp_seq=1 ttl=64 time=0.118 ms
64 bytes from 2001:db8:1:10::10: icmp_seq=2 ttl=64 time=0.074 ms
64 bytes from 2001:db8:1:10::10: icmp_seq=3 ttl=64 time=0.070 ms
ubuntu@kontejnermvlb1:~$ ping6 2001:db8:1:10::20
PING 2001:db8:1:10::20(2001:db8:1:10::20) 56 data bytes
64 bytes from 2001:db8:1:10::20: icmp_seq=1 ttl=64 time=0.058 ms
64 bytes from 2001:db8:1:10::20: icmp_seq=2 ttl=64 time=0.064 ms
64 bytes from 2001:db8:1:10::20: icmp_seq=3 ttl=64 time=0.070 ms
```

Obrázek 5.22: *Ping mezi kontejnery*

Nyní vidíme, že kontejnery se mezi sebou vidí a umí spolu komunikovat. Tyto kontejnery ale nejsou připojeny do sítě, kterou vytvořilo LXC. Aby se tyto kontejnery mohli dostat mimo macvlan sít, je potřeba využít možnost mít více rozhraní na jedné virtuální síťové kartě. Abychom mohli komunikovat s externí sítí, je potřeba vytvořit další kontejner, který bude mít více rozhraní a umožní přenos dat do macvlan sítě. K tomu si vytvoříme nový kontejner s názvem kontejnerdmz, který bude mít dvě virtuální síťová rozhraní.

- První je VETH síťové rozhraní spojené s lxcbr0 mostem - díky tomu s ním můžeme komunikovat z hosta, na kterém je spuštěn daný kontejner.
- Druhé je MAC VLAN síťové rozhraní ve stavu most, spojené s lxcbr1, takže kontejner může komunikovat s kontejnerem mvlb1 a kontejnerem mvlb2 kontejnery. Připomínám, že tyto dva kontejnery nejsou dostupné z hosta pomocí síťového rozhraní. Lze se k nim ale vždy připojit pomocí příkazu lxc-console.

Díky tomu získáme kontejner k ovládání z hosta a přes něj pak můžeme přistupovat na naše dva macvlan kontejnery. Vytvořím tedy nový kontejner s názvem kontejnerdmz.

```
NAME           STATE   AUTOSTART GROUPS IPV4          IPV6
kontejnerdmz   STOPPED 0         -      -             -
kontejnerempty STOPPED 0         -      -             -
kontejnermvlb1 RUNNING 0         -      10.0.5.3      2001:db8:1:10::10
kontejnermvlb2 RUNNING 0         -      10.0.5.4      2001:db8:1:10::20
kontejnerveth  RUNNING 0         -      10.0.3.167    2001:db8:1:0:216:3eff:fe7d:aa1
```

Obrázek 5.23: *Vytvoření kontejneru dmz*

Ověřím si, jestli jsou obě virtuální síťová rozhraní nastavena správně. Zajímá mě hlavně, jestli jsem dostal řádnou IPv6 adresu z radvd a jestli můžu komunikovat s kontejnery pro macvlan pomocí pevných adres.

```
ztr@ubuntu:~$ sudo lxc-ls --fancy
NAME           STATE   AUTOSTART GROUPS IPV4                IPV6
kontejnerdmz   RUNNING 0         -      10.0.3.92, 10.0.5.5 2001:db8:1:0:ca1:5bff:fe5c:c69c
kontejnerempty STOPPED 0         -      -                   -
kontejnermvlb1 RUNNING 0         -      10.0.5.3            2001:db8:1:10::10
kontejnermvlb2 RUNNING 0         -      10.0.5.4            2001:db8:1:10::20
```

Obrázek 5.24: *Výpis spuštěného kontejneru dmz*

Z obrázku je viditelné, že náš dmz kontejner obdržel jak IPv4 od LXC DHCP, tak statickou IPv4 pro komunikaci s macvlan kontejnery a k tomu dostal IPv6 od programu radvd, který rozděluje IPv6 adresy pro virtuální most lxcbr0.

```
ztr@ubuntu:~$ ping6 2001:db8:1:0:ca1:5bff:fe5c:c69c
PING 2001:db8:1:0:ca1:5bff:fe5c:c69c(2001:db8:1:0:ca1:5bff:fe5c:c69c) 56 data bytes
64 bytes from 2001:db8:1:0:ca1:5bff:fe5c:c69c: icmp_seq=1 ttl=64 time=0.101 ms
64 bytes from 2001:db8:1:0:ca1:5bff:fe5c:c69c: icmp_seq=2 ttl=64 time=0.090 ms
64 bytes from 2001:db8:1:0:ca1:5bff:fe5c:c69c: icmp_seq=3 ttl=64 time=0.078 ms
```

Obrázek 5.25: *Zkouška komunikace z hostu na kontejner dmz*

Nyní ověříme funkčnost IPv6 protokolu pomocí odeslání ICMP zprávy na kontejnerveth pomocí společného mostu lxcrb0.

```
ubuntu@kontejnerdmz:~$ ping6 2001:db8:1:0:216:3eff:fe7d:aa1 -c 3
PING 2001:db8:1:0:216:3eff:fe7d:aa1(2001:db8:1:0:216:3eff:fe7d:aa1) 56 data bytes
64 bytes from 2001:db8:1:0:216:3eff:fe7d:aa1: icmp_seq=1 ttl=64 time=0.074 ms
64 bytes from 2001:db8:1:0:216:3eff:fe7d:aa1: icmp_seq=2 ttl=64 time=0.082 ms
64 bytes from 2001:db8:1:0:216:3eff:fe7d:aa1: icmp_seq=3 ttl=64 time=0.084 ms
```

Obrázek 5.26: *Ping6 z kontejnerdmz na kontejnerveth přes lxcbr0*

Nyní vidíme, že z hosta můžeme s kontejnerem komunikovat na adrese přes rozhraní lxcbr0 a naopak nemůžeme komunikovat z hosta s rozhraním lxcbr1. To je chování, které očekáváme. Ještě ověření spojení přes rozhraní lxcbr0 na Google dns. Bohužel, můj poskytovatel internetového připojení zatím nepoužívá IPv6, tudíž můžu zkoušet dotazy do externí sítě pouze pomocí IPv4 ICMP zpráv.

```
ubuntu@kontejnerdmz:~$ ping 8.8.8.8 -c 3
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=53 time=17.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=53 time=16.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=53 time=25.3 ms
```

Obrázek 5.27: *Dosah na WAN síť*

Toto byla ukázka jak spojit macvlan kontejnery s externí sítí a jak k nim přistupovat pomoci dmz kontejneru a tím zabezpečit přístup k těmto macvlan kontejnerům. Toto užití je dobré používat tehdy, chceme-li izolovat podsíť od nadřazené sítě a přitom si zachovat možnost přístupu pro služby, které se mají na tyto kontejnery dostat.

5.5.1 Veth bridge

Toto zapojení je mi velmi sympatické, protože dokáže kontejner vystavit do lokání sítě a následně nám umožňuje práci na něm, jako na běžném PC v lokální síti. Vztahují se na něj stejná pravidla směrování jako by to byl fyzický počítač. S tímto nastavením je možné na kontejnery přistupovat z vnější sítě bez nutnosti rezervovat si fyzickou síťovou kartu pro sebe, jako to dělá například nastavení Phys, které budu rozebírat později. V ideální síti, kde již všechny počítače používají IPv6 adresy bychom na tento virtuální stroj přistupovali jenom pomocí IPv6 adresy. Momentálně, kdy je přechod na IPv6 pořád neúplný, musíme použít nástroj jako Cisco Any Connect, kterým se můžeme pomocí IPv4 adresy připojit na hraniční směrovač a poté, co od něj dostaneme IPv6, můžeme s tímto kontejnerem pracovat na IPv6 adresách.

Nejprve si připravíme nový síťový most, do kterého připneme naši fyzickou síťovou kartu. Jak jsem již řekl, toto nastavení si nerezervuje NIC pro sebe, ale umožní ji sdílet s ostatními virtuálními stanicemi pomocí nového mostu br0, který vytvoříme. Otevřeme si tedy konfigurační soubor fyzických síťových rozhraní na našem hostu a vytvoříme v něm nové rozhraní most br0. Tomuto mostu řekneme, že druhá strana spojení bude prováděna přes fyzické rozhraní enp2s0.

```
auto br0
iface br0 inet dhcp
        bridge_ports enp2s0
        bridge_stp off
        bridge_fd 0
        bridge_maxwait 0
```

Obrázek 5.28: *Nastavení pro br0 most*

Vytvoříme si tedy kontejner a použijeme pro něj následující síťové nastavení na kontejneru vethbr0, Po spuštění kontejneru bychom měli dostat IPv4 adresu a IPv6 adresu ze směrovače, který je umístěn ve vnější lokální síti.

```
ztr@ubuntu:~$ sudo lxc-ls --fancy
NAME             STATE   AUTOSTART GROUPS IPV4          IPV6
kontejnerdmz     STOPPED 0         -      -             -
kontejnerempty   STOPPED 0         -      -             -
kontejnermvlb1   RUNNING 0         -      10.0.5.3      2001:db8:1:10::10
kontejnermvlb2   RUNNING 0         -      10.0.5.4      2001:db8:1:10::20
kontejnerphys    STOPPED 0         -      -             -
kontejnerveth    RUNNING 0         -      10.0.3.167    -
kontejnervethbr0 RUNNING 0         -      192.168.0.100 2001:db8:0:1:216:3eff:fe16:9d70
```

Obrázek 5.29: *Spuštění kontejneru s přístupen do vnější LAN*

Můžeme vidět, že naše nastavení je správné, protože jsme obdrželi potřebné adresy, které nám neudělil radvd, ani lxcbr0. Nyní zkusíme dostupnost kontejneru.

```
C:\Windows\System32>tracert 2001:db8:0:1:216:3eff:fe16:9d70

Tracing route to 2001:db8:0:1:216:3eff:fe16:9d70 over a maximum of 30 hops

  1    <1 ms    <1 ms    <1 ms  2001:db8:0:1:216:3eff:fe16:9d70

Trace complete.

C:\Windows\System32>tracert 192.168.0.100

Tracing route to 192.168.0.100 over a maximum of 30 hops

  1    <1 ms    <1 ms    <1 ms  192.168.0.100

Trace complete.
```

Obrázek 5.29: *Ping na kontejner z vnější LAN*

Jak můžeme vidět, ping na kontejnery z lokální fyzické sítě byl úspěšný.

5.6 Macvlan

Dalším druhem sítě je macvlan. Tato síť je specifická tím, že dokáže použít jedno fyzické síťové rozhraní a vytvořit na něm několik virtuálních síťových rozhraní s odlišnými MAC adresami, které jsou k těmto virtuálním rozhraním přiřazeny.

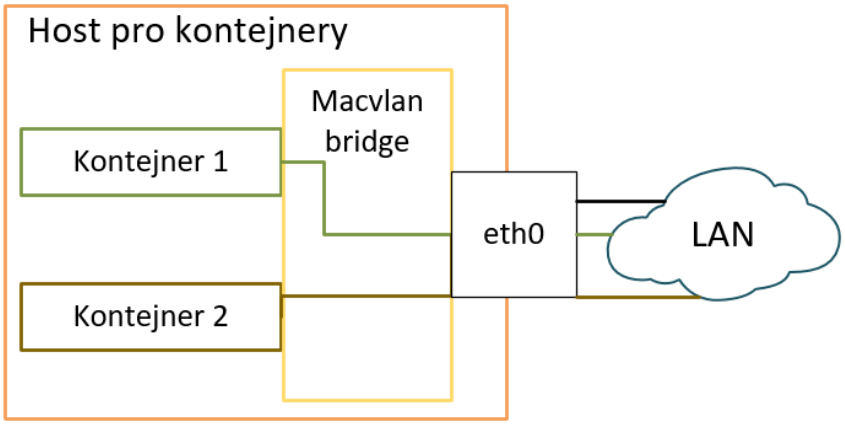
Tomu se říká many-to-one mapování. Linuxové VLANy mají schopnost použít jedno síťové rozhraní a namapovat ho do několika virtuálních sítí, což poskytne one-to-many mapování – jedno síťové rozhraní a více VLANů v jednom trunku. Macvlany mapují vícero síťových rozhraní do jednoho síťového rozhraní. Pokud chceme, můžeme kombinovat Linuxové VLANy s macvlany. Umožňují také každému nakonfigurovanému podřízenému zařízení využívat tří různých stavů.

5.6.1 Privátní

Zařízení nemá umožněno komunikovat s žádným nadřazeným zařízením, což znamená, že veškeré příchozí pakety na podřízené zařízení virtuálního rozhraní budou zahozeny, pokud jejich zdrojová MAC adresa odpovídá MAC VLAN rozhraní. To zakazuje právě komunikaci mezi jednotlivými MAC VLANy.

5.6.2 VEPA

Tento mód spojuje pakety virtuálních strojů na serveru předtím, než je výsledný tok přenesen do přepínače. Při použití VEPA předpokládáme, že nadřazený most vrací všechny rámce, kde zdroj a cíl jsou místní pro MAC VLAN port. To znamená, že daný most je nastaven na takzvané reflective relay. Tento stav se také nazývá hairpin mode a musí být podporován na přepínači určeném k odchozímu toku dat, nikoliv přímo v Linuxovém jádru. Takto směrujeme veškerý tok dat do přepínače v případě, že jsou data určena nám a přitom se spoléháme na přepínač, který data pošle zase zpátky. V tomto případě je izolace zajištěna právě daným přepínačem, nikoliv přímo Linuxovým jádrem.



Obrázek 5.30: *Macvlan bridge s použitím VEPA*

VEPA taktéž izoluje komunikaci mezi jednotlivými kontejnery, ale jen do doby, než máme odesílací přepínač, který je nakonfigurován jako reflective relay – v takovém případě lze kontejnery adresovat přímo. To má za následek, že tok dat určený odlišné MAC VLAN na stejném rozhraní (například na stejném mostu) se bude přenášet po lince dvakrát. Poprvé to bude, když bude paket opouštět rozhraní, na kterém je přepínáno a podruhé, když se paket bude zase vracet přes stejné rozhraní. V praxi může tento způsob ovlivnit rychlost komunikace fyzického zařízení (přepínače) a taktéž omezí rychlost vnitřní MAC VLAN sítě na rychlost fyzického přepínače. Jelikož VEPA vyžaduje speciální přepínač s konfigurací reflective relay, toto řešení zde nebudu testovat. Bohužel, tento přepínač nemám k dispozici.

5.6.3 **Vlan**

Vlan je vlastně virtuální síť, která umí rozdělit síťové zařízení na několik broadcastových domén. Abychom mohli vytvořit virtuální síť, která nám bude označovat pakety daným vlan id, musíme nastavit síť na kontejneru. Konfigurace této sítě je na obrázku 5.31.

```
# Network configuration
lxc.network.type = vlan
lxc.network.link = enp2s0
lxc.network.flags = up
lxc.network.vlan.id = 10
lxc.network.hwaddr = 00:16:3e:bd:f4:04
```

Obrázek 5.31: *Konfigurační soubor pro Vlan*

Nyní vytvoříme vlan rozhraní na hostu a spojíme jej s již existujícím rozhraním na hostu enp2s0. Rozhraní enp2s0 je označení fyzické síťové karty na našem hostu.

```
ztr@ubuntu:~$ sudo ip link add name vlan20 link enp2s0 type vlan id 20
ztr@ubuntu:~$ sudo ip link list vlan20
6: vlan20@enp2s0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFA
ULT group default qlen 1000
    link/ether 00:1e:8c:7a:db:59 brd ff:ff:ff:ff:ff:ff
```

Obrázek 5.32: *Vytvořené vlan rozhraní*

Vytvořili jsme nové rozhraní, které bude značkovat odchozí data pomocí VLAN identifikátorem číslo 20. Ověříme si, jestli je toto rozhraní nastaveno správně.

```
ztr@ubuntu:~$ sudo ip -d link show vlan20
6: vlan20@enp2s0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFA
ULT group default qlen 1000
    link/ether 00:1e:8c:7a:db:59 brd ff:ff:ff:ff:ff:ff promiscuity 0
    vlan protocol 802.1Q id 20 <REORDER_HDR> addrgenmode eui64
```

Obrázek 5.33: *Ověření konfigurace vlan*

Nyní toto rozhraní pošleme na kontejner vlan a zkontrolujeme, jestli bylo toto rozhraní přesunuto.

```
ztr@ubuntu:~$ ps faux|grep -A 1 kontejnervlan
ztr      23656  0.0  0.0  14224   972 pts/3    S+   09:14   0:00          |   |   \_ grep --color=auto -A 1 kontejnervlan
ztr      22615  0.0  0.0  22448  5060 pts/17   Ss+  09:05   0:00          |   \_ bash
root     22635  0.0  0.0  54492  4116 ?        Ss   09:05   0:00          \_ [lxc monitor] /var/lib/lxc kontejnervlan
root     22644  0.0  0.0  37224  5320 ?        Ss   09:05   0:00          |   \_ /sbin/init
ztr@ubuntu:~$ sudo ip link set vlan20 netns 22644
ztr@ubuntu:~$ ip l l vlan20
Device "vlan20" does not exist.
```

Obrázek 5.34: *Přesun vlan rozhraní z hosta na kontejner*

Rozhraní bylo úspěšně přesunuto. To nám potvrdil poslední řádek obrázku, kde nám host říká, že nenalezl dané rozhraní. Nyní se podíváme do kontejneru vlan, kde by se mělo toto rozhraní nyní nacházet.

```
ubuntu@kontejnervlan:~$ sudo ip link list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
 group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
5: eth0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP m
ode DEFAULT group default qlen 1000
    link/ether 00:16:3e:bd:f4:04 brd ff:ff:ff:ff:ff:ff link-netnsid 0
6: vlan20@if2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT
 group default qlen 1000
    link/ether 00:1e:8c:7a:db:59 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

Obrázek 5.35: *Úspěšné přenesení vlan rozhraní do kontejneru*

Naše VLAN byla přesunuta na kontejner vlan. Spojení z hosta se nyní změnilo z vlan20@enp2s0 na vlan2@if2 což znamená, že rozhraní pro vlan20 síť se spojilo s rozhraním na hostu, které má číslo 2. V podstatě jsme přenesli obsluhu tohoto rozhraní ze samotného hosta na kontejner.

Nesmíme zapomenout, že pokud provádíme toto zapojení na stroji, kde běží jiné kontejnery, které používají fyzické rozhraní enp2s0, bude jejich spojení s okolním světem nedostupné, protože jsme přesunuli IP adresy na odlišné rozhraní. To jsme si ukázali výpisem aktuálních rozhraní na hostu. Pokud chcete toto nastavení zkoušet s aktivními kontejnery, doporučuji zapojit další síťovou kartu. Zde jsme si zároveň dokázali, že nelze mít všechny konfigurace sítě pro LXC na jedné fyzické síťové kartě.

Užitečnost VLAN přichází tehdy, když potřebujeme začít naši síť rozšiřovat a chceme si udržet strukturu sítě kvůli bezpečnosti, nebo kvůli propustnosti sítě. Virtuální síťové rozhraní kontejneru můžeme skrýt za fyzické síťové rozhraní našeho hosta, který je zapojen do fyzické sítě. Taktéž bych doporučil toto virtuální nastavení používat jako možnost testování chování těchto sítí v reálném provozu.

Jedním ze zajímavých případů použití VLAN by mohlo být využití například použití na přepínači, který by měl třeba 8 portů. Mohli bychom nakonfigurovat porty 1-7 jako porty pro VLAN, kdy každá VLAN by měla jeden port na přepínači. Samotného hosta bychom potom připojili na port 8, který by sloužil jako trunk. Každý kontejner na tomto hostu by poté mohl využívat daný fyzický port a mohl by komunikovat na konkrétní VLAN. U tohoto řešení bych se ale poté obával, že by se mohl host stát úzkým hrdlem pro zbytek sítě.

Jenom bych upozornil, že u tohoto nastavení nelze mít více kontejnerů se stejným VLAN id spojených do jednoho fyzického rozhraní. To je ale dáno tím, na jakém principu VLAN pracují. Pokud bychom chtěli dosáhnout více kontejnerů se stejným VLAN id na jednom rozhraní, je potřeba použít způsobu připojení přes most. Tento způsob obsahuje vytvoření mostu, který se jmenuje br10 pro VLAN s ID 10.

```
sudo ip link add name vlan10 link eth0 type vlan id 10
```

Poté je nutno připojit zařízení vlan10 do mostu br10 a použít tento most pro kontejnery VLAN se značkou 10. Tyto kontejnery se vytváří se síťovým způsobem veth a spojení most je s br10.

5.7 Phys

Tento síťový způsob by si měl po konfiguraci převzít fyzickou síťovou kartu hostujícího stroje a připojit ji na kontejner. Takové rozhraní by mělo být přístupné na stejné síti, na které existovalo původní rozhraní - v podstatě izolujeme fyzické rozhraní na hostu podobně, jako jsme to provedli u způsobu veth.

Vytvoříme si nový kontejner s názvem kontejnerphys a provedeme konfiguraci jeho síťového nastavení. Nesmíme zapomenout nastavit výchozí bránu. Ta by měla odpovídat výchozí bráně nastavené na síťovém zařízení, ze kterého přejímáme síťové rozhraní. V našem

případě to je přepínač v domácí síti s IPv4 adresou 192.168.0.1. Bohužel, v tomto případě se mi nepodařilo zapojení zapojit pro IPv6, tak budu pracovat pouze s IPv4 adresami.

```
# Network configuration
lxc.network.type = phys
lxc.network.link = enp2s0
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:cf:95:8c
lxc.network.ipv4 = 190.168.0.111
lxc.network.ipv4.gateway = 192.168.0.1
```

Obrázek 5.36: *Konfigurační soubor pro Phys*

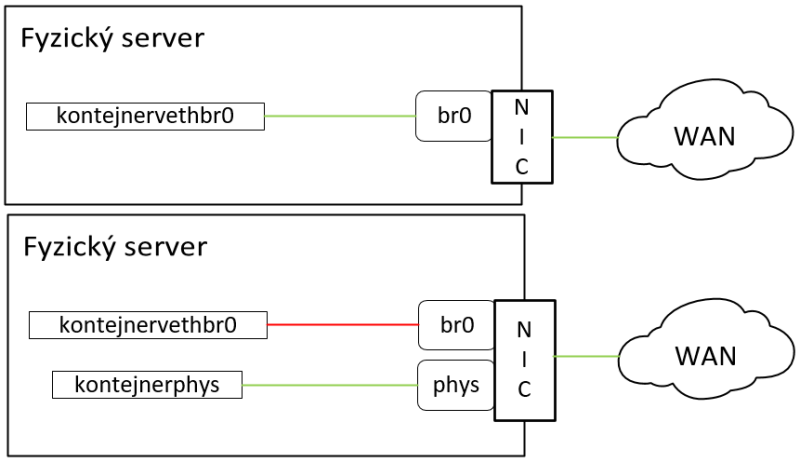
Kontejner si spustíme a přitom budeme pozorovat dostupnost kontejneru, který toto rozhraní používal ve svém br0 v kapitole 5.2.3. K tomuto jsem si spustil ping na počítači v lokální síti a používám jej na ověření dostupnosti kontejnerů z externí sítě. Nyní jsem konkrétně udělal ping na kontejner s veth bridge nastavením, který je nakonfigurován pro chování, jako by byl v lokální síti a tudíž musí i v této lokální síti dostupný.

```
Reply from 2001:db8:0:1:216:3eff:feed:cace: time<1ms
Reply from 2001:db8:0:1:216:3eff:feed:cace: time<1ms
Reply from 2001:db8:0:1:216:3eff:feed:cace: time<1ms
Request timed out.
Request timed out.
Request timed out.
```

Obrázek 5.37: *Nedostupnost ostatních kontejnerů*

Jak je patrné, tak po spuštění kontejneru se síťovým nastavením phys a odtrhnutím fyzického rozhraní enp2s0 došlo k přerušení veškeré komunikace ostatních kontejnerů, které používaly toto fyzické rozhraní ke komunikaci se sítí mimo našeho hosta. Z toho vyplývá, že není možné používat phys síťové nastavení tam, kde již máme použito třeba veth bridge nastavení, nebo tam, kde je fyzické síťové rozhraní užíváno například pro vlan nastavení. Tento problém by se ale dal vyřešit přidáním další síťové karty do fyzického hosta.

Na obrázku je stav, kdy vethbr0 kontejner může používat fyzickou síťovou kartu pro přístup do vnější sítě do té doby, než se spustí kontejner s nastavením phys, který si rezervuje celou fyzickou síťovou kartu pro sebe. V ten okamžik dojde k přerušení komunikace z kontejneru vethbr0 a začne rezervovaná komunikace mezi kontejnerem phys a fyzickou síťovou kartou.

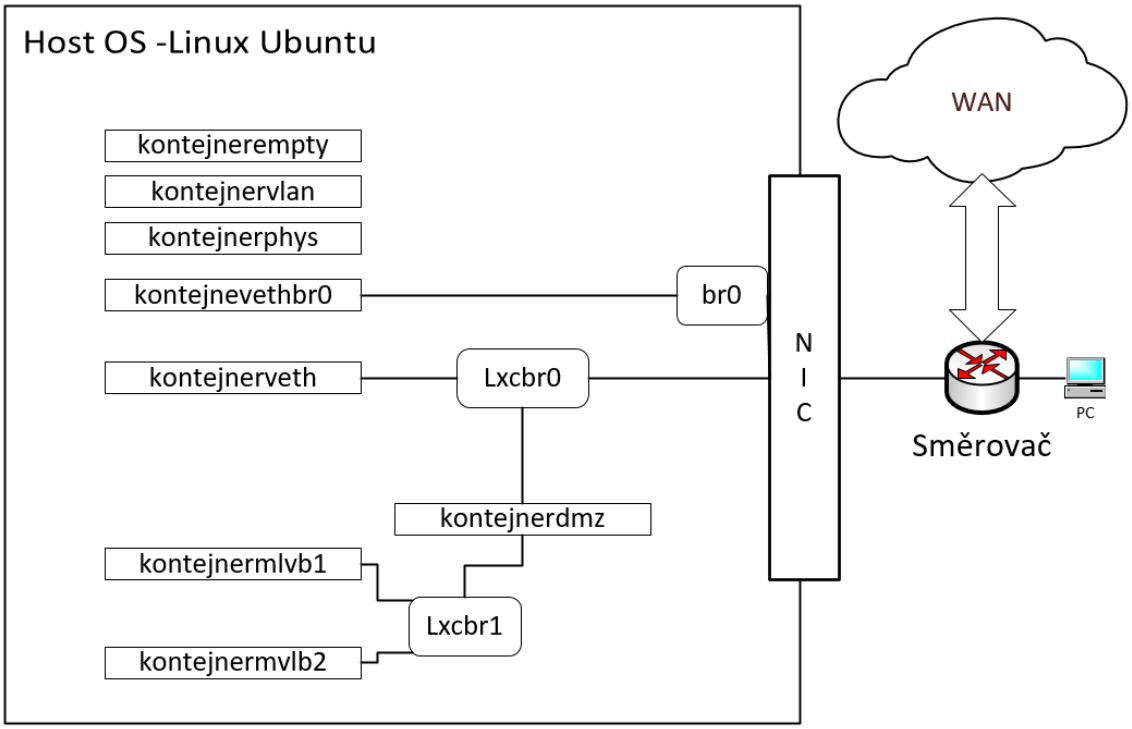


Obrázek 5.38: *Přerušení komunikace kontejneru vethbr0*

Toto nastavení virtuální sítě je vhodné tehdy, pokud chceme používat na kontejneru fyzickou síťovou kartu serveru a pokud potřebujeme být viditelní z LAN sítě bez dalšího nastavování.

5.8 Celková struktura práce

Pokusím se zde shrnout, jak to funguje na našem hostu a které kontejnery jsme vytvořily. V obrázku nejsou některé kontejnery zapojeny do NIC, protože to není paralelně možné, bez použití více než jedné NIC.



Obrázek 5.39: *Celkový pohled na naši virtuální síť*

Na obrázku 5.3.9 můžeme vidět, jak to vypadá uvnitř našeho hosta. Postupně jsme vytvořili několik kontejnerů s různým nastavením. Každé nastavení se chová individuálně dle parametrů, které jsme nastavili na virtuálním síťovém rozhraní kontejneru.

Kontejner pro empty nastavení je pouze kontejner s rozhraním loopback. Kontejner není vystaven do sítě a lze na něj přistupovat pouze pomocí lxc konsole. Kontejner ve stavu Phys není na obrázku zapojený do NIC, protože ho nelze paralelně spustit s jinými kontejnery, které používají NIC. Jak jsme si v práci ukázali, při spuštění kontejneru Phys dojde k přivlastnění NIC a ostatní kontejnery ztratí možnost komunikace se světem. Kontejner Vlan na obrázku taky nemáme zapojený, protože by nastala stejný situace, jako s kontejnerem Phys. Navíc na zapojení kontejneru Vlan je vyžadován speciální přepínač, který pracuje s technologií reflective relay.

Kontejner veth je příklad, jak lze k virtuálním stanicím přistupovat, jako bychom byli v lokální síti. Při tomto typu zapojení se vytváří spojení, které má na začátku i na konci síťové rozhraní. Při zapojení veth na hostu nejsou kontejnery přímo dostupné zvenku, ale kontejner je schopen vyslat dotaz do venkovní sítě.

Dalším zapojením bylo zapojení macvlan bridge. Pro toto zapojení jsme si vytvořili rozhraní lxcbr1. Na toto rozhraní jsme připnuli dva kontejnery, které nejsou schopny komunikace s okolním světem. Tyto kontejnery jsou tak izolované od zbytku sítě. Abychom tyto kontejnery mohli vystavit do sítě, vytvořili jsme si další kontejner s názvem dmz, který měl dvě síťová rozhraní a byl schopen jak komunikace s kontejnery přes lxcbr1, tak komunikace s ostatními kontejnery pomocí druhé rozhraní, které jsme na něm vytvořili.

Abychom mohli kontejner vystavit do lokální sítě a dále s ním pracovat ze sítě, jako s plnohodnotným PC, vytvořili jsme si rozhraní br0, do kterého jsme přidali NIC. K tomuto rozhraní jsme připnuly kontejner, který měl nastavení veth, ale byl spojený přímo s NIC pomocí mostu br0. Díky tomu se nám pak podařilo dosáhnout získání IP adres ze směrovače, který byl již v lokální síti.

6 Výkonnostní testování

K testování propustnosti jsem použil program iperf, který umí simulovat tok dat na daná fyzická i virtuální síťová rozhraní. V měření jsem chtěl dokázat, že virtuální síťové karty mají propustnost závislou na výkonu počítače a jeho komponent. U nevirtualizovaných strojů, je tato propustnost omezena fyzickým rozhraní síťové karty a aktivních síťových prvků. Tento test nepokrývá reálné rychlosti zápisu dat na fyzická úložiště (HDD, SSD).

```
ztr@ubuntu:~$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.0.115 port 5001 connected with 192.168.0.200 port 62716
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec   112 MBytes  93.9 Mbits/sec
[  5] local 192.168.0.115 port 5001 connected with 192.168.0.200 port 62718
[  5]  0.0-10.0 sec   107 MBytes  89.6 Mbits/sec
[  4] local 192.168.0.115 port 5001 connected with 192.168.0.200 port 62720
[  4]  0.0-10.0 sec   112 MBytes  94.0 Mbits/sec
[  5] local 192.168.0.115 port 5001 connected with 192.168.0.200 port 62721
[  5]  0.0-10.0 sec   112 MBytes  93.6 Mbits/sec
[  4] local 192.168.0.115 port 5001 connected with 192.168.0.200 port 62724
[  4]  0.0-10.0 sec  91.0 MBytes  76.3 Mbits/sec
[  5] local 192.168.0.115 port 5001 connected with 192.168.0.200 port 62729
[  5]  0.0-10.0 sec   105 MBytes  87.6 Mbits/sec
[  4] local 192.168.0.115 port 5001 connected with 192.168.0.200 port 62733
[  4]  0.0-10.0 sec  89.1 MBytes  74.7 Mbits/sec
[  5] local 192.168.0.115 port 5001 connected with 192.168.0.200 port 62735
[  5]  0.0-10.0 sec  90.9 MBytes  76.2 Mbits/sec
[  4] local 192.168.0.115 port 5001 connected with 192.168.0.200 port 62740
[  4]  0.0-10.0 sec   113 MBytes  94.2 Mbits/sec
```

Obrázek 6.1: Ukázka testování propustnosti ze strany iperf serveru

Na obrázku 6.1 vidíme výpis z testu iperf na straně hosta. Zde se jedná o fyzický host, na kterém jsou spuštěny naše kontejnery. Abychom mohli provést tento test, je potřeba mít nainstalovaný program iperf na obou stranách měřené linky. To znamená, že měřená virtuální stanice má iperf spuštěný v režimu server a poté spustíme na jiné stanici iperf v režimu klient a odesíláme data na server.

Příkaz pro spuštění iperf jako serveru na Linux:

```
Sudo iperf  -s
```

Testování na straně klienta obnáší spuštění programu iperf v režimu klient. To lze provést pomocí příkazu:

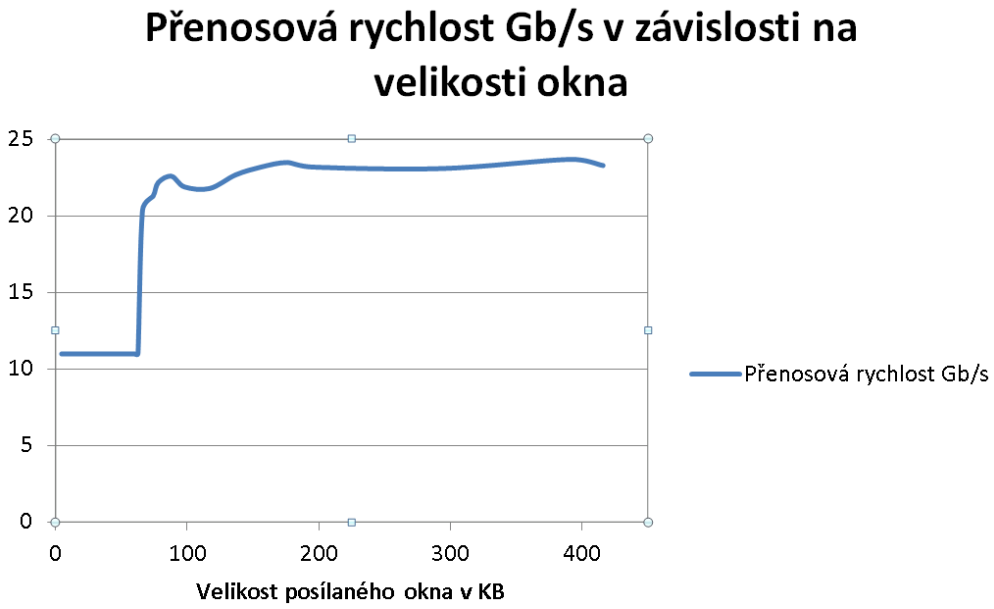
```
iperf -c cilovaIPadresa
```

Na obrázku 6.2 můžeme vidět provedení příkazu z klientské strany. Parametr -w nám dává možnost změnit velikost posílané okna.

```
ztr@ubuntu:~$ sudo iperf -c 10.0.3.167 -w 10000
------------------------------------------------------------
Client connecting to 10.0.3.167, TCP port 5001
TCP window size: 19.5 KByte (WARNING: requested 9.77 KByte)
------------------------------------------------------------
[  3] local 10.0.3.1 port 59914 connected with 10.0.3.167 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  13.5 GBytes  11.6 Gbits/sec
ztr@ubuntu:~$ sudo iperf -c 10.0.3.167 -w 30000
------------------------------------------------------------
Client connecting to 10.0.3.167, TCP port 5001
TCP window size: 58.6 KByte (WARNING: requested 29.3 KByte)
------------------------------------------------------------
[  3] local 10.0.3.1 port 59916 connected with 10.0.3.167 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  13.5 GBytes  11.6 Gbits/sec
ztr@ubuntu:~$ sudo iperf -c 10.0.3.167 -w 80000
------------------------------------------------------------
Client connecting to 10.0.3.167, TCP port 5001
TCP window size:  156 KByte (WARNING: requested 78.1 KByte)
------------------------------------------------------------
[  3] local 10.0.3.1 port 59918 connected with 10.0.3.167 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  28.3 GBytes  24.3 Gbits/sec
```

Obrázek 6.2: *Ukázka iperf testu z pohledu klienta*

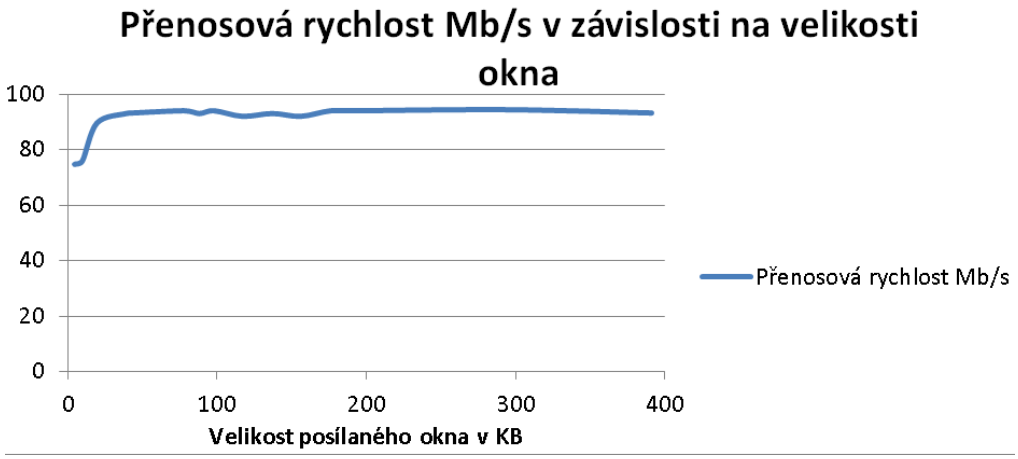
Na obrázku 6.3 vidíme, že již základní propustnost virtuálního stroje s výchozí velikostí posílaného okna je nad 11 Gb za sekundu. To je velmi slušný výsledek. To, co mě ale při testování překvapilo, bylo, že když jsem zkoušel různá nastavení velikosti okna, mohl jsem dosahovat vyšších přenosových rychlostí.



Obrázek 6.3: *Graf propustnosti síťového rozhraní mezi kontejnery*

Nejvyšší rychlost jsem dosáhl při velikosti okna 195 KB. Rychlost jsem zaznamenal téměř 24 Gb za sekundu, což je velmi krásná rychlost. Pokud bychom měli dostatečně rychlé úložiště, které by zvládalo zapisovat touto rychlostí, byla by virtualizovaná sít jednoznačně nejlepší volbou pro zálohování a kopírování dat. Samozřejmě, že i fyzická síťová karta může dosahovat rychlosti až 52 Gb za sekundu. Bohužel, cena těchto karet je příliš vysoká a tudíž jsou tyto karty nedostupné pro obyčejného uživatele. Navíc v dnešní době ještě domácí uživatelé nepřenáší po své síti takové množství dat, aby se rychlosti takových síťových karet všeobecně využily.

Další měření jsem prováděl s vědomím, že přenosové rychlosti budou limitovány rychlostí fyzických síťových prvků. Bohužel, porty na mém přepínači mají rychlost pouze 100 Mb za sekundu.



Obrázek 6.*4*: *Graf propustnosti síťového rozhraní mezi LAN a kontejnerem*

Jak je vidět na obrázku 6.4, rychlost byla opravdu velmi limitována. S malou velikostí okna byla přenosová rychlost lehce pod 80 Mb za sekundu, ale se zvětšování okna již nemohlo dojít rychlejšímu přenosu dat, protože to prostě fyzická síťová karta nepropustí. Byl bych rád, kdybych měl k dispozici síťové karty o rychlosti alespoň 10 Gb za sekundu, na kterých by se dala lépe testovat závislost propustnosti na velikosti okna.

6.1 Měření cest

Abychom si ověřili, jak naše data opravdu tečou, prověřil jsem tyto cesty pomocí Windows příkazu:

```
tracert ipadresa
```

a pro Linux příkazem:

```
tracepath6 ipadresa
```

Nejprve jsem si prověřil základní cestu z fyzického hosta na fyzický PC ve vnější síti.

```
ztr@ubuntu:~$ tracepath6 2001:db8:0:1:695c:69b2:e793:6aae
 1?: [LOCALHOST]                                         pmtu 1500
 1:  2001:db8:0:1:695c:69b2:e793:6aae                      0.829ms reached
 1:  2001:db8:0:1:695c:69b2:e793:6aae                      0.773ms reached
     Resume: pmtu 1500 hops 1 back 1
```

Obrázek 6.5: *Tracepath6 na PC ve vnější síti*

Jako další test jsem prověřil cestu z kontejneru, který má přístup do internetu. Zde mě překvapilo, že cesta opravdu jenom přes jeden skok.

```
ubuntu@kontejnervethbr0:~$ tracepath6 2001:db8:0:1:695c:69b2:e793:6aae
 1?: [LOCALHOST]                                         pmtu 1500
 1:  2001:db8:0:1:695c:69b2:e793:6aae                      0.281 ms reached
 1:  2001:db8:0:1:695c:69b2:e793:6aae                      0.205 ms reached
     Resume: pmtu 1500 hops 1 back 1
ubuntu@kontejnervethbr0:~$
```

Obrázek 6.6: *Tracepath6 z kontejneru na vnější PC*

Jako poslední měření cesty ukážu cestu mezi kontejnery, které společně používají rozhraní lxcbr0. U tohoto rozhraní bych předpokládal, že cesta bude mít jeden skok navíc v podobně výchozí brány na rozhraní lxcbr0. Toto rozhraní má totiž fyzickou adresu a měla by se nám v měření ukázat. Test jsem provedl mezi kontejnerem s veth rozhraním a kontejnerem určeným pro dmz.

```
ubuntu@kontejnerdmz:~$ tracepath6 2001:db8:1:0:216:3eff:fe7d:aa1
 1?: [LOCALHOST]                                         pmtu 1500
 1:  2001:db8:1:0:216:3eff:fe7d:aa1                        0.104 ms reached
 1:  2001:db8:1:0:216:3eff:fe7d:aa1                        0.034 ms reached
     Resume: pmtu 1500 hops 1 back 1
ubuntu@kontejnerdmz:~$
```

Obrázek 6.7: *Tracepath6 mezi kontejnery na lxcbr0*

Na obrázku 6.7 vidíme, že tracepath nám ukazuje pouze jeden skok. To je něco, co jsem nečekal. Myslel jsem si, že by se tam měla zobrazit i výchozí brána rozhraní lxcb0, jelikož oba tyto virtuální kontejnery toto rozhraní používají a získávají skrze něj IP adresy.

Závěr

Hlavním cílem této diplomové práce bylo popsat problematiku virtualizace sítí. V této práci jsem psal o technologiích VEB a kontejnerové virtualizaci. Snažil jsem se co nejlépe popsat způsoby komunikace virtuálních síťových zařízení dle různých nastavení a s použitím protokolu IPv6.

Během své práce jsem si prověřil jednotlivá síťová nastavení pro LXC kontejnerového správce, se kterým jsem pracoval po celou dobu práce. V práci jsem se snažil vysvětlit, proč je používání kontejnerů značně úspornější k fyzickým zdrojům serverů a jak lze s kontejnery manipulovat a zajišťovat tak jejich zálohy, popřípadě udržovat automatické vyvažování zátěže na více fyzických serverech. Práce obsahuje detailní nastavení sítě pro LXC kontejnery a taktéž se zabývá zkoumáním reálných cest dat v dané virtuální síti. Snažil jsem se potvrdit, nebo vyvrátit domněnky, které v oblasti síťové virtualizace kontejnerů panovaly. Během práce jsem se naučil samotnou práci s těmito síťovými rozhraními a prakticky ověřil, jak tyto způsoby síťové virtualizace implementovat na dané fyzické prostředí.

Jedním z hlavních cílů této práce bylo najít nejvhodnější síťové nastavení kontejneru tak, aby byl tento kontejner přístupný z vnější sítě pomocí IPv6 adresy. Toho jsem nakonec dosáhl pomocí kombinace možnosti vytvoření virtuálního síťového mostu mezi fyzickou síťovou kartou a virtuální síťovou kartou kontejneru pomocí síťového způsobu veth. Taktéž jsem zjistil, které síťové způsoby lze kombinovat s použitím jedné fyzické síťové karty a způsoby, ke kterým je vhodné více mít vice než jednu fyzickou síťovou kartu.

Tato diplomová práce je přínosem hlavně pro studenty a lidi, kteří budou s těmito technologiemi později pracovat, protože virtualizace je v dnešní době jeden z podstatných prvků při návrhu počítačových serverů a sítí. Pomocí této práce si může každý zájemce vytvořit kontejnery a spojovat je v rámci virtuálního prostředí pomocí různých síťových nastavení. Myslím si, že nejatraktivnější na virtuálních sítích jsou přenosové rychlosti. Ty se pohybují v násobku desítek proti standartním síťovým kartám v dnešních počítačích.

Pro další vývoj této práce bych doporučil práci na fyzické stanici s více než jednou fyzickou síťovou kartou, aby se dalo vyzkoušet zapojení všech síťových nastavení zároveň. Dalším zlepšením by bylo použití rychlejších fyzických síťových karet. Tím bychom získali perfektní ukázku možností virtuálních síti v kombinaci se sítěmi fyzickými.

Použitá literatura

[1] Boyce Gregory: Linux Networking Cookbook, Packt Publishing, 2016, ISBN-13: 978-1785287916.

[2] Cobbaut Paul: Mastering Linux - Networking, Samurai Media Limited, 2016, ISBN-13: 978-9888406210

[3] IEEE 802.1Q-2014 - Bridges and Bridged Networks

[4] Kangovi Sachidananda: Peering Carrier Ethernet Networks, Morgan Kaufmann, 2016, ISBN-13: 978-0128053195

[5] Ethernet Virtual Bridging Automation Use Cases. [online]. Dostupné z: http://www.itc22.com/fileadmin/ITC22_files/Ethernet_Virtual_Bridging_Automation_Use_Cases_-_final.pdf

[6] Ko Mike, Recio Renato: Virtual Ethernet Bridging. [online]. http://www.ieee802.org/1/files/public/docs2008/new-dcb-ko-VEB-0708.pdf

[7] Gajdos Milos: Exploring LXC Networking. [online.] http://containerops.org/2013/11/19/lxc-networking/

[8] Pham Tim: LXC Advanced Networking - Exposing Containers to the Network. [online.] http://www.bonsaiframework.com/wiki/display/bonsai/5.1+LXC+Advanced+Networking+-+Exposing+Containers+to+the+Network